SOFTWARE MAINTAINABILITY STUDY:

A FRAMEWORK

By

QI SUN

Bachelor of Science

University of Petroleum

Shangdong, China

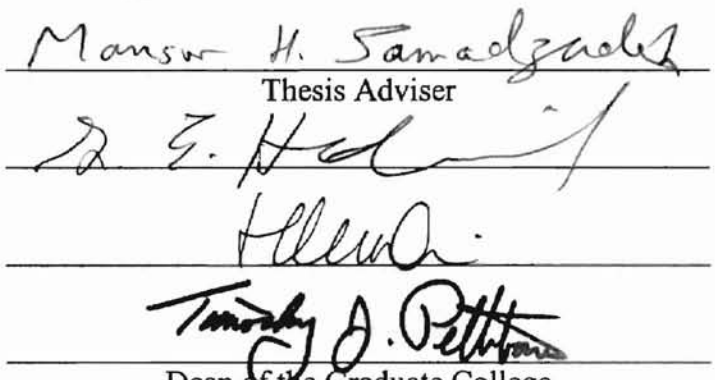
1993

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December 2002

SOFTWARE MAINTAINABILITY STUDY:

A FRAMEWORK

Thesis Approved



Thesis Adviser

Dean of the Graduate College

PREFACE

In the study of software maintainability, the sizing of the software is of critical importance. In this thesis work, several direct and indirect methods of sizing are introduced. They include lines of code, function points, and object points. After obtaining information about software size, the software development and maintenance effort was calculated by using the COCOMO II or COSMOS models. Maintainability index is an evaluation standard for software maintenance. One method for calculating the maintainability index of software is by using the sizing of the software. Another important factor in assessing maintainability is by using risk analysis, like technical risk, schedule risk, etc. In this thesis work, three risk evaluation methods, SRAM, CFHRAM, and CCRAM, were used for software risk assessment. Two case studies were provided: one is a real time display system with more than 100,000 lines of code and the other is an operating system simulator with several thousand lines of code.

The main objectives of this thesis was to build a comprehensive but easy to follow framework for the study of software maintainability. A software maintainability study can consist of four steps. The first step is to use lines of code, function points, or object points to size the software. The second step is to use COCOMO II or COSMOS to calculate the software development and maintenance effort. The third step is to use McCabe's Cyclomatic number and Halstead's measures to calculate the software maintainability index. The last step is to choose SRAM, CFHRAM, or CCRAM to analyze software risk according to the characteristics of the software. Two cases studies

showed that the framework has sufficient flexibility and can be followed effectively and easily.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to and thank my thesis advisor Dr. Mansur H. Samadzadeh, who motivated me by his valuable instruction and example throughout my thesis research work. Dr. Samadzadeh gave generously of his valuable time and expertise for correcting my work and rendering helpful suggestions.

I also extend my sincere appreciation to Dr. G. E. Hedrick and Dr. H. K. Dai for their advice and willingness to serve on my graduate committee.

Also, I would like to express my gratitude to my wife, Su Zhang. Without her support, understanding, and encouragement, I would not have been able to finish the study.

Finally, I want to thank all my friends who were mentally with me all the time.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER I

INTRODUCTION

In software engineering, there are many areas and topics such as Process Management, Configuration Management, Requirement Engineering, Software Testing, Software Measurement, Software Cost Estimation, Software Reliability, and Software Maintenance. The focus of this thesis work was Software Maintenance. A lot of progress has been made in the area of Software Maintainability and many books and papers can be found in this area. Software sizing refers to estimating the size of software. Software sizing is important for cost, effort, and schedule estimation.

This thesis work discusses several direct and indirect methods of software sizing (lines of code, function points, object points, and Cyclomatic complexity), the software development and maintenance efforts estimation (COCOMO II and COSMOS), a maintainability index calculation method, and several risk evaluation methods (SRAM, CDHRAM, and CCRAM). Two case studies are provided: one is a real time system with more than 100,000 lines of code, and the other is an operating system simulator with several thousand lines of code.

The rest of this thesis is organized as follows. Chapter II of this thesis provides a literature review on Software Sizing (function points, object points, and Cyclomatic complexity). The classification of lines of code is discussed in Chapter III. In Chapter IV,

two software development and maintenance efforts estimation methods (COCOMO II and COSMOS) are described. Software maintainability is discussed is Chapter V. In Chapter VI, three risk evaluation methods (SRAM, CDHRAM, and CCRAM) are described. Two case studies are provided in Chapter VII. Finally, the summary of this thesis and some areas of future work are presented in Chapter VIII.

CHAPTER II

SOFTWARE SIZING

The accuracy of software cost, effort, and schedule estimation depends directly or indirectly on software size estimation. There are a number of approaches for sizing software for the purpose of addressing the problem of cost, effort, and schedule estimation. These approaches include lines of code, function points, and object points. These methods attempt to capture, measure, and quantify some aspect of the bulk of software. Among these, the lines of code (LOC) approach is the one most widely used. Najberg stated, "although alternatives to lines of code have been proposed over the years, it appears that lines of code will remain the standard for cost estimation purposes" [Najberg 84]. LOC will be discussed separately in Chapter III. Function points, Cyclomatic complexity, and Halstead's measures are briefly explained in the following three sections of this chapter.

2.1 Function Points (FP)

2.1.1 Definition

The function point concept, initially published in 1983 [Albrecht and Gaffney 83], is a popular approach to software size estimation. As the name implies, function points measure the functionality visible to the user and delivered to the user. The following

explanations about function points are based on the article titled "The Software Cost Modeling System (COSMOS)" by Henry [Henry 98].

Functionality is divided into two kinds [Henry 98]: data functionality and transaction functionality. Data Functionality is supplied through logical groups of data that are read or maintained. Transaction Functionality is supplied through processes provided by the software. The functionality delivered is the sum of these two kinds of functionalities.

2.1.1.1 Data Functionality

The files, database tables, objects, and other information storage entities provide data functionality [Henry 98]. The physical data sources are grouped into conceptually complete or "logical" groups of data. There are two kinds of data groups in function point analysis as explained below.

- Logical data groups on that are maintained within the application boundary (i.e., written by the application). These are called Internal Logical Files (ILFs).
- Logical data groups on that provide information to the application, but are maintained by another application. These are called External Interface Files (EIFs).

2.1.1.2 Transaction functionality

The processes furnished by an application provide transaction functionality [Henry 98]. These processes group the individual actions of the program into self-consistent, conceptually complete operations that process or produce information and leave the software and its data in a stable, consistent state. These are "elementary processes" or the smallest units of activity that are meaningful to the user. Smaller units of activity, and activity that does not leave the application in a stable state, are not counted. The processes within the application are identified based on the information

that enters and exits the application. Data that cross the application boundary signal the operation of transaction functions. There are three kinds of transaction functions in function point analysis: external input, external output and external inquiry, as explained below.

External Input (EI): Identified by data or control information that comes into the application to perform an elementary process. Control information is information used by the application to satisfy a business process requirement. Data that comes into the application is used to maintain an Internal Logical File. Typically, EI processes are used to add, change, or delete information.

External Output (EO): Identified by calculated or derived data that exit the application. Often EO processes are reports.

External Inquiry (EQ): Identified by an input/output combination that supplies a body of data in response to a request from outside the application. The data contains lookup information only, no calculated or derived data are supplied (otherwise the process is an External Output). The data supplied to application to perform the lookup does not maintain an Internal Logical File (otherwise the process is an External Input). Typically, EQ processes are requests to view information.

2.1.2 Complexity

Each function identified in the function point analysis (i.e., EI, EO, EQ, ILF, or EIF) is weighted according to its complexity.

For data functions (IFLs and EIFs), the complexity is determined and rated by the numbers of Record Element Types (RETs – distinct record formats) and the number of Data Element Types (DETs – distinct fields) contained within the logical group of data [Boehm et al. 98].

The complexity rankings given in Table 1 are used for the Internal Logical Files [Boehm et al. 98].

Table 1. Complexity of Internal Logical Files

	1 to 19 DET	20 to 50 DET	51 or more DET
1 RET	Low	Low	Average
2-5 RET	Low	Average	High
6 or more RET	Average	High	High

where DET = Number of data element types (fields)
RET = Number of Record Element Types (subgroups of Internal Logical Files based on logical/user view of the data)

The complexity rankings given in Table 2 are used for the External Interface Files [Boehm et al. 98].

Table 2. Complexity of External Logical Files

	1 to 19 DET	20 to 50 DET	51 or more DET
1 RET	Low	Low	Average
2-5 RET	Low	Average	High
6 or more RET	Average	High	High

where DET = Number of data element types (fields)
RET = Number of Record Element Types (subgroups of External Logical Files based on logical/user view of the data)

For transaction functions (EIs, EOs, and EQs), the complexity is determined and rated by the number of File Types Referenced (FTRs – number of ILFs and EIFs used by

the process) and by the number of Data Element Types (DETs – distinct fields) added, changed, deleted, or produced in outputs.

The complexity rankings given in Table 3 are used for the External Inputs [Boehm et al. 98].

Table 3. Complexity of External Inputs

	1 to 4 DET	5 to 15 DET	16 or more DET
0-1 FTR	Low	Low	Average
2 FTR	Low	Average	High
3 or more FTR	Average	High	High

where DET = Number of data element types (fields)
FTR = Number of Internal Logical Files maintained or referenced plus number of External Interface files referenced during processing of the External Input

The complexity rankings given in Table 4 are used for the External Outputs [Boehm et al. 98].

Table 4. Complexity of External Outputs

	1 to 5 DET	6 to 19 DET	20 or more DET
0-1 FTR	Low	Low	Average
2-3 FTR	Low	Average	High
4 or more FTR	Average	High	High

where DET = Number of data element types (fields)
FTR = Number of Internal Logical Files referenced plus number of External Interface files referenced during processing of the External Output

The following procedure is used to rank the complexity of the External Inquiries:

a. Compute the input side of the External Inquiry (just like an External Input).

b. Compute the output side of the External Inquiry (just like an External Output).

c. Use the higher of the two results.

The complexity weights for the different functions are given below [Boehm et al. 98] in Table 5. These weights are used to calculate the final number of function points.

Table 5. Complexity of Different Functions

		LOW	AVERAGE	HIGH
Internal Logical File	ILF	7	10	15
External Interface File	EIF	5	7	10
External Input	EI	3	4	6
External Output	EO	4	5	7
External Inquiry	EQ	3	4	6

2.2 Cyclomatic Complexity (CC)

Cyclomatic complexity has its foundation in graph theory and is a useful and widely used software metric that provides a quantitative measure of the logical complexity of a program [Pressman 01]. The value of Cyclomatic complexity is the number of independent paths in a program, and provides an upper bound for the number of tests that must be conducted to ensure all statements have been executed at least once.

An independent path in a program is any path through the program that introduces at least one new set of executable statements or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been visited before the path is defined [Pressman 01].

Cyclomatic complexity can be calculated in a number of different ways, three of which are mentioned below.

I. The number of regions of the flow graph of a program, provided that the flow graph is planar.

II. $V = E - N + 2$, where V is the Cyclomatic complexity of the program, E is the number of edges, and N is the number of nodes of the program flow graph.

III. $V = P + 1$, where V is the Cyclomatic complexity of the program and P is the number of binary predicate nodes contained in the program flow graph.

2.3 Halstead's Complexity Measures

Halstead's measures are based on four scalar numbers derived directly from a program's source code [Halstead 77].

n1 = number of distinct operators

n2 = number of distinct operands

N1 = total number of operators

N2 = total number of operands

From these numbers, five measures are derived, as shown in Table 6 below.

Table 6. Halstead's Measures

Measure	Symbol	Formula
Program Length in Tokens	N	$N = N_1 + N_2 = n_1 \log n_1 + n_2 \log n_2$
Program Vocabulary	n	n = n1 + n2
Volume	V	V = $N \log n$
Difficulty	D	$D = (n_1/2)(N_2/n_2)$
Effort	E	E = DV

CHAPTER III

LINES OF CODE (LOC)

There are a large number of models that have been built either directly or indirectly based on LOC and its estimation. In this thesis, the lines of code models are divided into six categories: analogy models, regression models, expert judgment models, models based on function point, parametric models, and the Software Engineering Institute (SEI) model. A brief description is given below for each model.

3.1 Analogy Models

A model or technique in this category of estimating LOC is characterized by comparing a program with a similar program of known size or similar programs of known sizes. Reifer [Reifer 86] gives a simple equation as follows:

$$S = F \times (SizeofSimilarPackages)$$

where S is LOC and F is a factor determined by experience and/or policies.

3.2 Regression Models

Models in this category are similar to the Analogy Models in that they both use historical data. But Regression Models do not make direct comparisons; rather, they perform regression analysis on the historical data and derive size estimation equation from different factors [Ikatura and Takayanagi 82]. A common equation is as follows:

$$Y = \sum_{i=0}^{n} C_i X_i$$

where Y is LOC, i is number of program characteristics, C_i are the coefficients and X_i represent program characteristics. C_i are determined through the result of regression analysis.

3.3 Expert Judgment Models

In these techniques, several experts are consulted for their opinions regarding software size. One of most basic expert judgment models is Program Evaluation and Review Technique (PERT) equation [Boehm 81] [Reifer 86], which is as follows.

$$E = \frac{a + 4m + b}{6}$$

where E is the expected size (LOC), a is the smallest possible size, m is the most likely size, and b is the largest possible size. The values of a, b, and m are determined by the experts.

3.4 Models Based on Function Points

The function point concept was introduced and discussed in Section 2.1. This section focuses on the correspondence between function points and lines of code. Typically, the function point count, which is determined by a detailed inspection of the specification and design documents, will have to be converted to lines of code in the implementation language (assembly, higher order languages, fourth-generation languages, etc.) in order to assess the relative conciseness of implementation per function point. Earlier effort on finding the correspondence between function points and LOC was based on a regression analysis equation between function point count and LOC. One such

equation is the correlation equation derived after studying 48 COBOL and PL / I programs in data processing applications [Albrecht and Gaffney 83].

$$S = 53.2F + 12773$$

where S is LOC and F is the number of function points.

Typically, the unadjusted function point (UFP) count will have to be converted to source lines of code in the implementation language (assembly, higher order language, fourth-generation language, etc.) in order to assess the relative conciseness of implementation per function point. More recent research has focused on finding conversion factors between function points and LOC. One conversion factor set [Jones 91] to translate a function point into equivalent LOC, and vice versa, is shown in Table 7.

Table 7. Converting a Function Point to Lines of Code [Jones 91]

Language	LOC/UFP
Ada	71
AI Shell	49
APL	32
Assembly	320
Assembly (Macro)	213
ANSI/Quick/Turbo Basic	64
Basic - Compiled	91
Basic - Interpreted	128
C	128
C++	29
ANSI Cobol 85	91
Fortran 77	105

Forth	64
Jovial	105
Lisp	64
Modula	280
Pascal	91
Prolog	64
Report Generator	80
Spreadsheet	6

However, there is no universal standard for the conversion factor between LOC and UFP. It varies depending on software type (database software, operating systems software, numerical software, etc.), language, algorithm, and developer experience. Table 8 [Fischman 00] lists another set of conversion factors between LOC and UFP.

Table 8. Converting a Function Point to Lines of Code [Fischman 00]

Language	LOC/UFP
Ada	73
Java	49
APL	59
Assembly	320
Assembly (Macro)	213
Basic	58
Delphi	51
SQL	19
C	61

C++	59
ANSI Cobol 85	61
Fortran 77	58
Access	23
PL/1	71
Lisp	58
Modula	61
Pascal	91
Prolog	61
Visual C++	54
Spreadsheet	6

3.5 Parametric Models

These models use input parameters consisting of numerical or descriptive values of selected program attributes. Most parametric models can be calibrated, i.e., suitable values for one or more input parameters can be determined using historical data.

One example of a parametric model used for software size estimation is the RCA PRICE Sizer model [RCA PRICE Systems 87]. This model requires over 15 inputs that describe the characteristics of the program being estimated including the number of alphanumeric and graphic displays, input and output streams, control states. This model uses a factor called SICAL that can be calibrated to a user's organization if sufficient historical data is available.

3.6 SEI Model

The Software Engineering Institute, at Carnegie Mellon University in Pittsburgh, PA, has developed a checklist for measuring LOC as part of a system of definition checklists, report forms, and supplemental forms to support measurement definitions [Park 92] [Jones 99].

Table 9 below shows a portion of the definition checklist as it is being applied to support the development of a model. Each checkmark in the "Include" column identifies a particular statement type or attribute included in the definition, and the opposite holds for the "Exclude" column. Other parts of the definition checklist clarify statement attributes for usage, delivery, functionality, replications, and development status.

Table 9. Definition Checklist for Source Statement Counts [Park 92]

Measurement unit	Physical source lines						
	Logical source statements			4			
Statement type	Definition	4	Data Array			Include	Exclude
When a line or statement contains more than one type, classify it as the type with the highest precedence.							
1 Executable	Order of precedence				1	4	
2 Non-executable							
3 Declaration					2	4	
4 Compiler directive					3	4	
5 Comment							
6 Counts on lines from their sources					4		4
7 Counts on lines with other source code					5		4

8 Banners and non-blank spacers					6		4
9 Blank (empty) comments					7		4
10 Blank lines					8		4
How produced	Definition	4	Data Array			Include	Exclude
1 Programmed						4	
2 Generated with source code generators							4
3 Converted with automated translators						4	
4 Copied or reused without change						4	
5 Modified						4	
6 Removed							4
Origin	Definition	4	Data Array			Include	Exclude
1 New work: no prior existence						4	
2 Prior work: taken or adapted from							
3 A previous version, build, or release						4	
4 Commercial, off-the-shelf software (COTS), other than libraries							4
5 Government furnished software (GFS), other than reuse libraries							4
6 Another product							4
7 A vendor-supplied language support library (unmodified)							4
8 A vendor-supplied operating system or utility (unmodified)							4
9 A local or modified language support library or operating system							4

10 Other commercial library			4
11 A reuse library (software designed for reuse)		4	
12 Other software component or library		4	

Some tools that implement this checklist are available on the Internet, a famous one is CodeCount from the University of Southern California (USC).

CHAPTER IV

SOFTWARE COST ESTIMATION

The estimation of software development cost has been the focus of much research over the past 20 years. In software engineering, cost estimation is closely related to effort and schedule estimation. As a result, cost estimation and effort estimation are sometimes used interchangeably. Boehm classifies software cost estimation models as follows [Boehm 81].

1) Linear models or mathematical models that attempt to fit a simple line onto the observed data.
2) Multiplicative models that express effort as a product of constants with various cost drivers and their exponents.
3) Analytical models that usually express effort as a function that is neither linear nor multiplicative.
4) Tabular models that represent the relationship between cost drivers and development effort in a matrix form.
5) Composite models that use a combination of all or some of the aforementioned approaches.

Composite models have the advantage of being generic enough to represent a large class of situations. Their mathematical definitions make it easy to implement them on a computer. Some widely used composite models are COCOMO II [Boehm et al. 98],

COSMOS [Henry 98], and the RCA PRICE Sizer [RCA PRICE Systems 87]. In the following two sections, COCOMO II and COSMOS are discussed in some detail.

4.1 COCOMO II

The Constructive Cost Model II (COCOMO II) was devised by Barry Boehm and his associates [Boehm et al. 98] to produce parametric estimates of software project effort and schedule.

Boehm defined a hierarchy of models: basic, intermediate, and advanced. Basic COCOMO II is a static single-valued model that can be used to estimate development effort and cost as a function of the estimated number of source lines of code (LOC). Intermediate COCOMO II calculates development effort and cost as a function of program size in terms of estimated LOC and a subjective assessment of 17 "cost drivers". Advanced COCOMO II includes all intermediate features plus an assessment of each cost driver's impact on each phase of the system life cycle.

COCOMO II has two equations, one for computing effort and the other for computing schedule. In COCOMO II, the product of 17 cost driver rating values calibrates the effort calculation.

The effort estimation equations used in COCOMO II [Boehm et al. 98] are as follows.

$$PM = \prod_{i=1}^{17}(CD_i)\cdot A\cdot\left[\left(1+\frac{BRAK}{100}\right)\cdot Size\right]^{B} + \left(\frac{ASLOC\cdot\left(\frac{AT}{100}\right)}{ATPROD}\right) \qquad \text{Eq4.1-1}$$

where:

$$Size = KNSLOC + \left[KASLOC\left(\frac{100-AT}{100}\right)\cdot\frac{(AA+SU+0.4\cdot DM+0.3\cdot CM+0.3\cdot IM)}{100}\right]$$

Eq4.1-2

$$B = 0.91 + 0.01\sum_{j=1}^{5} SF_j \qquad \text{Eq4.1-3}$$

The symbols in the above equations are briefly explained below and will be further discussed in Chapter VII of Case Studies.

Symbol	Description
A	a constant, calibrated as 2.45
AA	assessment effort, the effort of estimate whether or to what extent an existing software can be reused
ASLOC	the number of source lines of code adapted from existing software used in developing the new product
AT	percentage of components that are automatically translated
BRAK	breakage: percentage of code thrown away due to requirement volatility
CD	cost driver
CM	percentage of code modified
DM	percentage of design modified
IM	percentage of integration and test modified
KASLOC	size of the adapted component expressed in thousands of adapted source lines of code
KNSLOC	size of component expressed in thousands of new source lines of code
PM	person months of estimated effort
SF	scale factors

SU software understanding (zero if DM = 0 and CM = 0)

The schedule estimation equations used in COCOMO II [Boehm et al. 98] are as follows.

$$TDEV = \left[3.67 \times \left(\overline{PM}\right)^{(0.28+0.2\times(B-1.01))}\right]\frac{SCED\%}{100} \qquad \text{Eq4.1-4}$$

where:

$$B = 0.91 + 0.01\sum_{j=1}^{5} SF_j \qquad \text{Eq4.1-5}$$

The symbols in the above equations are explained below.

Symbol	Description
PM	person Months of estimated effort (excluding the effect of the SCED effort multiplier)
SF	scale factors
TDEV	time to develop
SCED	schedule
SCED%	the compression/expansion percentage in the SCED cost driver

4.2 COSMOS

The Software Cost Modeling System (COSMOS) was developed by Henry [Henry 1998]. It combines Albrecht's function point analysis (International Function Point User Groups, IFUG standard), Boehm's COCOMO model [Boehm et al. 98], and Putnam's adaptation of the Rayleigh distribution model [Putnam 80]. The function point model of COSMOS is described below.

It should be noted that applying the COSMOS model does not help with function point count (Section 2.1) per se, a knowledgeable user still has to correctly identify EIs, EOs, EQs, ILFs, and EIFs of low, average, and high complexity. COSMOS can calculate the function point count (i.e., it can apply the complexity weights and the value adjustment factors) from a user's "raw" function counts and general system characteristic settings.

Unadjusted Function Point Count is in COSMOS is given as follows.

$$UFPC = (3 \times LEI) + (4 \times AEI) + (6 \times HEI) + (4 \times LEO) + (5 \times AEO) + (7 \times HEO) + (7 \times LILF) + (10 \times AILF) + (15 \times HILF) + (5 \times LEIF) + (7 \times AEIF) + (10 \times HEIF) + (3 \times LEQ) + (4 \times AEQ) + (6 \times HEQ)$$

where:

UFPC=Unadjusted function point count
LEI= Number of low external inputs
AEI =Number of average external inputs
HEI=Number of high external inputs
LEO=Number of low external outputs
AEO=Number of average external outputs
HEO=Number of high external outputs
LILF=Number of low internal logical files
AILF=Number of average internal logical files
HILF=Number of high internal logical files
LEIF=Number of low external interface files
AEIF=Number of average external interface files
HEIF=Number of high external interface files
LEQ=Number of low external inquiries
AEQ=Number of average external inquiries
HEQ=Number of high external inquiries

The Total Degree of Influence (TDI) is the sum of the selected complexity ratings for 14 project complexity traits, where the ratings are as follows.

None	0
Insignificant	1
Moderate	2
Average	3
Significant	4
Strong	5

The Value Adjustment Factor (VAF) is used to convert unadjusted function points into adjusted function points [Boehm et al. 98]. The VAF is computed from the Total Degree of Influence (TDI) as follows.

$$VAF = 0.65 + (0.01 \cdot TDI)$$

where:

VAF = Value Adjustment Factor
TDI = Total Degree of Influence

The Adjusted Function Point Count is the function point count as adjusted to compensate for environmental conditions or circumstances that impact software development [Boehm et al. 98]. The Adjusted Function Point Count is computed from the Unadjusted Function Point Count (UFPC) and the Value Adjustment Factor (VAF) as follows.

$$AFPC = UFPC \cdot VAF$$

where:

AFPC = Number of function points
UFPC = Function point count

VAF = Adjusted processing complexity

LOC is estimated from the adjusted function point count by using the selected programming language's "Conversion Factors" [Jones 91] as follows.

$$SLOC = AFPC \cdot LM$$

where:

	Source lines of code
AFPC =	Number of function points
LM =	Conversion Factors, a function point-to-LOC conversion constant for the selected language

One can estimate effort by simply multiplying an adjusted function point count by the productivity rate (FP/month) — if the productivity rate is known and where function point count measures the kind of work performed (i.e., user-defined functionality that is added, removed, changed, or converted).

CHAPTER V

MAINTAINABILITY

Measurement of software maintainability is of great interest in software engineering. One simple way to calculate maintainability is to use the empirical evidence available about distribution of maintenance tasks [Peters and Pedrycz 00]. Let c (corrective maintenance), a (adaptive maintenance), and p (perfective maintenance) have weights of 0.2, 0.25, and 0.55, respectively. Then maintainability can be measured using estimates of the average number of days required for each of the principal maintenance tasks as follows.

Maintainability = 0.2 (Avg # of days repairing code)+0.25 (Avg # of days adapting code) +0.55 (Avg # of days enhancing code)

The drawback of this method is that it depends on the knowledge of average days of various maintenance activities.

In an effort to qualify software maintainability better, several polynomial regression models have been defined [Oman and Hagemeister 92] [Oman and Hagemeister 94] [Zhou et al. 93]. A modified maintainability index was given by Lowther as either of the following equations [Zhou et al. 93].

I. Three-metric Maintainability Index (MI) equation

$$MI = 171 - 5.2\ln(\text{Avg V}) - 0.23\text{AvgV(g)} - 17.2\ln(\text{Avg LOC})$$

where *Avg V* is the average Halstead's Volume per module, *Avg V(g)* is the average Cyclomatic complexity per module, and *Avg LOC* is the average lines of code per module.

II. Four-metric MI equation

$$MI = 171 - 5.2\ln(\text{Avg E}) - 0.23\text{AvgV(g)} - 16.2\ln(\text{Avg LOC}) + 50\sin(\sqrt{2.4\text{perCM}})$$

where *Avg E* is the average Halstead Effort per module, *Avg V(g)* is the average Cyclomatic complexity per module, *Avg LOC* is the average lines of code per module, and *perCM* is the average percent of lines of comments per module.

To determine which equation is more suitable to a project, human judgment is very important. Generally speaking, if it is believed that the comments in the code significantly contribute to maintainability, then the four-metric MI equation is the better choice, otherwise the three-metric MI equation is more appropriate.

Coleman established two quality cutoffs for analyzing systems [Coleman 92]. A value above 85 indicates that the software is highly maintainable, a value between 85 and 65 suggests moderate maintainability, and a value below 65 indicates that the system is difficult to maintain.

CHAPTER VI

SOFTWARE RISK ANALYSIS

Risk in the context of software engineering is defined as uncertainties and factors that may cause late delivery, cost overrun, or lowered quality of a software product [Foo and Muruganantham 00].

There are a number of published models that evaluate the risk of a software project. Pressman discussed a method to evaluate risk using risk drivers [Pressman 01]. This approach was conceived following US Air Force's guidelines for software risk identification and abatement. The US Air force defined the major risk components as performance risk, cost risk, support risk, and schedule risk. This model however, does not have questions that bring out process related risks and is more suited for acquisition than development of software.

Another model, named Software Engineering Risk Model (SERIM) [Dale 1996], focuses on three risk elements: technical risk, cost risk, and schedule risk. Similar to the Risk Driver model above, SERIM is based on subjective probability. SERIM defines a hierarchical probability tree formulated by risk elements, risk factors, and risk metrics of decision alternatives. The model however, does not take into account the software complexity issues which play an important role in determining risk for a software project. It also does not account for issues related to the requirements document.

In this thesis, the case studies reported in the next chapter use three risk assessment models: Software Risk Assessment Model (SRAM) [Foo and Muruganantham 00], Cost Factor Heuristic Risk Assessment Model (CFHRAM) [Madachy 97], and Cyclomatic Complexity Risk Assessment Model (CCRAM) [McCabe and Watson 94]. The SRAM model is based on a list of questions concerning the nature of software, the development environment, developers, etc. The CFHRAM model is based on cost drivers. The CCRAM model is based on a program's Cyclomatic complexity. These three models are briefly explained in the following three sections.

6.1 Software Risk Assessment Model (SRAM)

SRAM considers the following nine critical risk elements: complexity of software, staff involved in the project, targeted reliability, product requirements, method of estimation, method of monitoring, development process adopted, usability of software, and tools used for development. A set of questions was carefully chosen for each of these elements with three choices of answers for each [Foo and Muruganantham 00]. The three possible answers are to be arranged in increasing order of risk.

For example, Software Complexity is one of the risk elements of software projects. The higher the complexity of the software, the higher is the risk. One of the questions used to assess the risk associated with the complexity of software and the choices for the answer are given below.

Q1. What is the function of the software to be developed?

a. Data processing software

b. Service software (Communication software)

c. System software

The risk assessor will pick one of the three choices based on the nature of the project and the actual situation. Choice a is assigned a risk rating of one, choice b is assigned a rating of two, and choice c is assigned a rating of three. In the above example, system software is considered the most complex among the three types of software, as it has to interact with the hardware and facilitate the operation of other types of software. Data processing software, on the other extreme, only deals with local data and hence is deemed to be the least complex of the three.

Let the nine risk element probabilities in the SRAM model be denoted by r_1, r_2, ..., r_9. As the nine risk elements have different degrees of impact on different types of software projects, different weights may be assigned to these elements when combining the risk element probabilities to derive an overall risk value for a project. Let the weights assigned to the elements be denoted by w_1, w_2, ..., w_9. The risk level R of the project is then computed as $w_1 r_1 + w_2 r_2 + ... + w_9 r_9$. If the maximum rating for all questions is 3 and the minimum rating is 1, the maximum value of R is given by $R_{max} = w_1 3 + w_2 3 + ... + w_9 3 = 3(w_1 + w_2 + ... + w_9)$ and the minimum value of R by $R_{min} = w_1 + w_2 + ... + w_9$. The overall risk level R may then be normalized as follows.

Normalized $R = R_n = (R - R_{min})/(R_{max} - R_{min})$

The value of R_n provides the risk level of the assessed project as a fraction between 0 and 1. R_n for a project with the lowest possible risk (no risk) is 0 and R_n for a project with the highest possible risk is 1. In SRAM model, this normalized value for a project is referred to as project risk.

The level of risk of a project in relation to quality, schedule, and cost can also be assessed separately based on the risk element probabilities obtained [Foo and Muruganantham 00]. This is done by assigning different weights to the probabilities

according to the impact of the associated risk elements on quality, schedule, and cost, respectively. The weighting proposed by Foo and Muruganantham [Foo and Muruganantham 00] is given in the Table 10. The normalized value R_n computed using these values for weight and the risk element probabilities, gives an indication of the level of risk of the project in relation to quality. The level of risk in relation to schedule and cost can be obtained in a similar fashion.

Table 10. Impact of Risk Element on Quality, Schedule, and COST

Risk Element	Quality	Schedule	Cost
Complexity	High	High	High
Staff	High	High	High
Reliability	High	Medium	Medium
Requirements	High	Medium	Medium
Estimation	Medium	Medium	Medium
Monitoring	Low	Medium	Low
Development Process	Medium	Medium	Medium
Usability	Medium	Low	Low
Tools	Low	Low	Low

If the risk level of a project assessed by SRAM is above 0.6, this indicates that it is a high-risk project. In such a case, it is strongly advised to discontinue or not to undertake the project [Foo and Muruganantham 00]. If the risk level of a project assessed by SRAM is between 0.36 and 0.6, it is necessary to reduce the likelihood and impact of the risk elements through immediate risk containment procedures. The project manager

should identify the weak areas that require urgent attention and resources. If the risk level of a project assessed by SRAM is below 0.36, the project is considered a low-risk project and the chances of success are high.

6.2 Cost Factors Heuristic Risk Assessment Model (CFHRAM)

In this model, risk impact or risk exposure is defined as the probability of loss multiplied by the cost of the loss [Madachy 97]. The equation given below is a quantitative risk-weighting scheme that accounts for the non-linearity of the assigned risk levels and cost multiplier data to compute the overall risks for each category and for the entire project.

$$risk = \sum_{j=1}^{C} \sum_{i=1}^{CR} RL_{ij} \times CDP_{ij}$$

where C is number of categories (i.e., personnel, technical, complexity), CR is number of category risks, RL is risk level, and CDP = (driver 1 cost driver) * (driver 2 cost driver) * ... * (driver n cost driver)

And, if the risk involves a schedule constraint,

CDP = ((schedule constraint cost driver) / (relative schedule)) × (driver 1 cost driver) × (driver 2 cost driver) × ... × (driver n cost driver)

6.3 Cyclomatic Complexity Risk Assessment Model

In the context of risk assessment, a common application of the Cyclomatic complexity (Section 2.2) is to compare it against a set of threshold values [McCabe and Watson 94]. One such threshold set is given in the table below.

Table 11. Threshold Values for Risk Assessment Using Cyclomatic Complexity

Cyclomatic Complexity	Risk Evaluation
1-10	A simple program, without much risk
11-20	More complex, moderate risk
21-50	Complex, high risk program
Greater than 50	Untestable program (very high risk)

While code is under development, it can be measured for Cyclomatic complexity to assess the inherent risk or risk buildup. Code complexity tends to increase as it is maintained over time. By measuring the Cyclomatic complexity before and after a proposed change, this buildup can be monitored and used to help decide how to minimize the risk associated with each change.

CHAPTER VII

CASE STUDIES

Two cases were used to conduct maintainability studies using the models discussed in Chapters IV, V, and VI.

The first case study was a commercial database management and Real Time Display (RTD) system that has over 100,000 lines of code. It was a comparatively complex system implemented in C++.

The second case study was an operating system simulation (OSS). This was a class project for a graduate-level operating systems course with two phases.

The quantitative information in the rest of this chapter (Sections 7.1 and 7.2) were obtained from the documentations and the codes for the RTD System and the OSS. The documentations included the requirements/specification documents, various manuals, and code documentation. Information collected from interviews with the designers, programmers, as well as users of the RTD System were also included. The author of this thesis was the programmer for the OSS, and the thesis adviser was the instructor of the Operating System (OS) course. The major tools used for extracting and collecting the quantitative data were CodeCount developed by University of Southern California Center for Software Engineering in 1998 (which can be downloaded at the URL http://sunset.usc.edu/research/CODECOUNT/index.html), and Understanding for C++ developed by Scientific Toolworks, Inc. (which can be obtained at the URL http://www.scitools.com/downloadc.shtml).

7.1 Real Time Display (RTD) System

The RTD System (this is a software used in the airline industry and air traffic control) had two parts, one was RTDI, and the other RTDII.

7.1.1 Function Points

The functionalities of RTDII were generated as follows.

Table 12. Functionality Counts of RTDII

EI (Low)	EO (Low)	EQ (Low)	ILF (Low)	ILF (High)	EIF (High)
44	28	5	12	1	2

The functionalities of RTDI were generated as follows.

Table 13. Functionality Counts of RTDI

EI (Low)	EO (Low)	EO (High)	EQ (Low)	EQ (High)	ILF (Low)	ILF (High)	EIF (Low)
76	174	2	19	2	16	20	18

The unadjusted function points of RTDII is 44*3 + 28*4 + 5*3 +12*7 + 15 + 2*10 = 378 and the unadjusted function points of RTDI is 76*3 + 174*4 + 2*7 + 19*3 + 2*6 + 16*7 + 20*15 + 18*5 = 1509. As explained below, These numbers were converted to function points to be used as a basis for estimation.

The conversion factor for the two subsystems for C++ is 29 [Jones 91] and 59 [Fischman 00] (Section 3.4), respectively. Due to the different natures of RTDII and RTDI, the conversion factor for RTDII and the conversion factor for RTDI should be different. RTDI is much more complex than RTDII, so a conversion factor of 55 LOC/UFP was used for RTDI and a conversion factor 45 LOC/UFP was used for RTDII.

The basis for the selection of these conversion factors was the experiments conducted by the author of this thesis (extracting different parts of RTDII and RTDI, size varying between several hundred and tens of thousand lines, and comparing with the hand counting of corresponding function points).

7.1.2 Software Cost Drivers

The software cost estimation model in Chapter IV adjusts the calculation of effort by an Effort Adjustment Factor (EAF). The EAF was derived from 17 cost driver attributes [Boehm et al. 98]. The cost drivers were grouped into four major categories: product, platform, personnel, and project attributes. Each of the attributes was rated based on its influence on project complexity using a 6-point scale ranging from very low to extra high (not all ratings are allowed for all attributes). Based on the rating, the corresponding effort multiplier of each cost driver was used to compute the EAF. The EAF is the product of the rating weights of the 17 cost drivers.

The following discussion and tables show the various cost drivers and the ranking of the RTD System against these cost drivers.

7.1.2.1 Product Factors

7.1.2.1.1 Required Software Reliability (RELY)

This is the measure of the extent to which the software must perform its intended function over a period of time [Boehm et al. 98]. If the effect of a software failure is only a slight inconvenience, then RELY is low. If a failure would risk human life, then RELY is very high.

Table 14. Required Software Reliability

	Very Low	Low	Nominal	High	Very High
RELY	slight in-convenience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life
Value	0.82	0.92	1.00	1.10	1.26
RTD Ranking					X

The RTD System is ranked Very High as its failure would represent potential risk to human life.

7.1.2.1.2 Database Size (DATA)

This measure attempts to capture the effect that large data requirements have on product development. The rating is determined by calculating the ratio D/P (explained below).

D/P=DataBaseSize(bytes) / ProgramSize(LOC).

DATA is rated as low if D/P is less than 10 and it is very high if it is greater than 1000 [Boehm et al. 98].

Table 15. Database Size

	Low	Nominal	High	Very High
Data	<10	10<=D/P <100	100 <= D/P <1000	D/P >= 1000
Value	0.90	1.00	1.14	1.28

RTD Ranking				X

The RTD System can have up to 100,000,000 pages (a page is the basic container object and display unit of RTDI; a page can contain an image, a button, or other objects). We assumed each page was 2 Kbytes (the average page size in one of the test sites appears to be larger than 2 Kbytes). Thus the estimated database size of the RTD System would be 20,000,000 Kbytes. The estimated LOC count of the RTD System was less than 150,000, thus for the RTD System we have D/P>1000. So the RTD System is ranked as Very High for database size.

7.1.2.1.3 Product Complexity (CPLX)

Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations [Boehm et al. 98]. In order to determine Product Complexity, we needed to select the area or the combination of areas that characterized the product or a subsystem of the product. The complexity rating is the subjective weighted average of these areas.

Table 16. Product Complexity

	Control Operations	Computational Operations	Device Dependent Operations	Data Management Operations	User Interface Management Operations
Very Low	Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IFTHENELSEs; simple module composition via procedure calls or simple scripts	Evaluation of simple expressions: e.g., *A=B+C*(DE)*	Simple read, write statements with simple formats	Simple arrays in main memory. Simple COTS-DB queries, updates	Simple input forms, report generators
Low	Straightforward nesting of structured programming operators, mostly simple predicates	Evaluation of moderate-level expressions: e.g., *D=SQRT(B**2-4.*A*C)*	No cognizance needed of particular processor or I/O device characteristics; I/O done at GET/PUT level	Single file subsetting with no data structure changes, no edits, no intermediate Files; moderately complex COTS-DB queries and updates	Use of simple graphic user interface (GUI) builders
Nominal	Mostly simple nesting; some intermodule control; decision tables; simple callbacks or message passing, including middleware supported distributed processing	Use of standard math and statistical routines; basic matrix/vector operations	I/O processing includes device selection, status checking, and error processing	Multi-file input and single file output; simple structural changes and simple edits; complex COTS-DB queries and updates	Simple use of widget set

High	Highly nested structured programming operators with many compound predicates; queue and stack control; homogeneous, distributed processing; single processor soft real time control	Basic numerical analysis: multivariate interpolation and ordinary differential equations; basic truncation and round off concerns	Operations at physical I/O level (physical storage address translations:; seeks, reads, etc.); optimized I/O overlap	Simple triggers activated by data stream contents; complex data restructuring	Widget set development and extension; simple voice I/O, multimedia
Very High	Reentrant and recursive coding; fixed-priority interrupt handling; task synchronization, complex callbacks, and heterogeneous distributed processing; single-processor hard real time control	Difficult but structured numerical analysis: near singular matrix equations and partial differential equations; simple parallelization.	Routines for interrupt diagnosis and servicing, masking; communication line handling; performance intensive embedded systems	Distributed database coordination; complex triggers; search optimization	Moderately complex 2D/ 3D, dynamic graphics and multimedia

Extra High	Multiple resource Scheduling with dynamically changing priorities; microcode-level control; distributed hard real time control	Difficult and unstructured numerical analysis: highly accurate analysis of noisy and stochastic data; complex parallelization	Device timing dependent Coding and micro-programmed operations; performance-critical embedded systems	Highly coupled, dynamic relational and object structures; natural language data management	Complex multimedia and virtual reality

Table 17. RTD Ranking of Database Size [Boehm et al. 98]

	Very Low	Low	Nominal	High	Very High	Extra High
CPLX Value	0.73	0.87	1.00	1.17	1.34	1.74
RTD Ranking			X			

For the RTD System, all five areas were used. The system used multiple resource scheduling with dynamically changing priorities, so the control operations was rated Extra High (weight 5).

Since RTD System is primarily a data retrieval and not a computation system, a 0 was assigned to the computational operations, so the computational operations is rated Very Low (weight 0).

The RTD System contained I/O processing including device selection, status checking, and error processing, so the device dependent operations was rated Nominal (weight 2).

Data management in the RTD System was very basic and simple array operations, so the operations was rated Very Low (weight 0).

The RTD System had simple use of the widget set, so the user interface management operations was rated Nominal (weight 2).

The average weight of the RTD System was (5 + 0 + 2 + 0 + 2)/5 = 1.8, which was near the value of Nominal. Thus the Product Complexity of the RTD System was considered Nominal.

7.1.2.1.4 Required Reusability (RUSE)

This cost driver accounts for the additional effort needed to construct components intended for reuse in the current or future projects [Boehm et al. 98]. This effort is consumed due to creating a more generic design for the software, more elaborate documentation, and more extensive testing to ensure that the components are ready for use in other applications.

Table 18. Required Reusability

	Very Low	Low	Nominal	High	Very High	Extra High
RUSE		none	Across project	across program	across product line	Across multiple product lines
Value		0.95	1.00	1.07	1.15	1.24
RTD Ranking					X	

The RTD System design is based on UML design models/approaches and it is implemented in Visual C++ (an object oriented language), hence it is ready for reuse in future versions. Also, the system has used some original C code from RTDI_A (The previous and much smaller version of the RTD system). Consequently, the level of Required Reliability is considered to be Very High.

7.1.2.1.5 Documentation Match to Life-Cycle Needs (DOCU)

Several software cost models have a cost driver for the level of required documentation. The rating scale for the DOCU cost driver is evaluated in terms of the suitability of the project's documentation to its life cycle needs [Boehm et al. 98]. The rating scale goes from Very Low (few life-cycle needs covered) to Very High (extensive coverage of life-cycle needs).

Table 19. Documentation Match to Life-Cycle Needs

	Very Low	Low	Nominal	High	Very High
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered	Right-sized to life-cycle needs	Excessive for life-cycle needs	Extensive life-cycle need coverage
Value	0.81	0.91	1.00	1.11	1.23
RTD Ranking					X

Since the RTD System is very complex and possibly is going to be very widely used (extrapolating from RTDI_A) in its 20-year expected life cycle, it requires extensive documentation coverage of life-cycle needs for training, operation, and maintenance. So the documentation match to life cycle needs (DOCU) would be rated as Very High.

7.1.2.2 Platform Factors

The platform factors refer to the target-machine complex of hardware and infrastructure software (virtual machine). Some additional platform factors were considered such as distribution, parallelism, embeddedness, and real-time operations, as explained in the following three subsections.

7.1.2.2.1 Execution Time Constraint (TIME)

This is a measure of the execution time constraint imposed upon a software system [Boehm et al. 98]. The rating is expressed in terms of the percentage of available execution time expected to be used by the system or subsystem consuming the execution time resource. The rating ranges from nominal (less than 50% of the execution time resource used) to extra high (95% of the execution time resource consumed).

Table 20. Execution Time Constraint

	Very Low	Low	Nominal	High	Very High	Extra High
TIME			<=50% use of available execution time	70%	85%	95%
Value			1.00	1.11	1.29	1.63
RTD Ranking						X

Since the RTD System is basically a real time system, it requires high tolerances and timing constraints in executing the displayed objects, among other things. The execution time constraint (TIME) of the RTD System was ranked as Extra High.

7.1.2.2.2 Main Storage Constraint (STOR)

This rating represents the degree of main storage constraint imposed on a software system or subsystem [Boehm et al. 98]. Given the remarkable increase in available processor execution time and main storage, one can question whether this constraint variable is still relevant. However, many applications continue to expand to consume whatever resources are made available, making such cost drivers still relevant. The rating ranges from nominal (less that 50%) to extra high (95%).

Table 21. Main Storage Constraint

	Nominal	High	Very High	Extra High
STOR	<=50% use of available storage	70%	85%	95%
Value	1.00	1.05	1.17	1.46
RTD Ranking	X			

The common hard disk capacity of each site's workstation for the RTD System was 8 Mbytes, and thus at most 5,000 pages could be created on each workstation. Considering the average size of a page, the hard disk capacity seemed adequate. Thus the main storage constraint of the RTD System was ranked as Nominal.

7.1.2.2.3 Platform Volatility (PVOL)

"Platform" is used here to mean the complex of hardware and software (OS, DBMS, etc.) that the software product calls on to perform its tasks. If the software to be developed is an operating system, then the platform is the computer hardware. If a

database management system is to be developed, as is the case with RTD system, then the platform is the hardware and the operating system. If a network text browser is to be developed, then the platform is the network, computer hardware, the operating system, and the distributed information repositories. The platform includes any compilers or assemblers supporting the development of the software system. This rating ranges from low (there is a major change every 12 months) to very high (there is a major change every two weeks) [Boehm et al. 98].

Table 22. Platform Volatility

	Low	Nominal	High	Very High
PVOL	major change every 12 mo.; minor change every 1 mo.	major: 6 mo.; minor: 2 wk.	major: 2 mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days
Value	0.87	1.00	1.15	1.30
RTD Ranking	X			

The development environment of the RTD System was not going to change more frequently than the hardware and the operating system, so the Platform Volatility of the RTD System was ranked as Low based on the available choices.

7.1.2.3 Personnel Factors

7.1.2.3.1 Analyst Capability (ACAP)

Analysts are personnel who work on requirements, high-level design, and detailed design. The major attributes that should be considered in this rating are analysis and design ability, efficiency and thoroughness, and the ability to communicate and cooperate. The rating should not consider the level of experience of the analyst by only

ranking his/her capability against the general population of analysts. Experience is considered a separate factor that was rated as AEXP (defined in Subsection 7.1.2.3.3). Analysts that fall in the 15th percentile of all phases of program development are rated very low, and those that fall in the 90th percentile are rated as very high [Boehm et al. 98].

Table 23. Analyst Capability

	Very Low	Low	Nominal	High	Very High
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile
Value	1.42	1.19	1.00	0.85	0.71
RTD Ranking					X

The analysts working on the RTD System had similar project experiences, so the analyst capability of the RTD System was ranked as Very High.

7.1.2.3.2 Programmer Capability (PCAP)

Current trends continue to emphasize the importance of highly capable analysts. However, the increasing role of complex COTS packages and the significant productivity leverage associated with programmers' ability to deal with these COTS packages (which the RTD System's development required), indicates a trend toward higher importance of programmer capability as well.

Evaluation should be based on the capability of the programmers as a team rather than as individuals. Major factors which should be considered in the rating were: ability,

efficiency and thoroughness, and the ability to communicate and cooperate. The experience of the programmer should not be considered here, rather the goal is only ranking his/her capability against the general population of programmers. Experience is considered a separate factor it was rated with AEXP (defined in Subsection 7.1.2.3.3). A very low rated programmer team averages its capability in the 15th percentile of all development activities, and a very high rated programmer team averages in the 90th percentile [Boehm et al. 98].

Table 24. Programmer Capability

	Very Low	Low	Nominal	High	Very High
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile
Value	1.34	1.15	1.00	0.88	0.76
RTD Ranking					X

Almost all of the programmers of the RTD System here continuously worked on the successive generations of the RTD System. The experience thus gained and the very low turnover rate in a competitive information technology job market resulted in assigning the programmer capability of the RTD team a Very High ranking.

7.1.2.3.3 Application Experience (AEXP)

The rating for this parameter is dependent on the level of application experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application [Boehm et al. 98]. A very low rating is for application experience of less than 2 months

with systems of the type being developed. A very high rating is for experience of 6 years or more.

Table 25. Application Experience

	Very Low	Low	Nominal	High	Very High
AEXP	2 months	6 months	1 year	3 years	6 years
Value	1.22	1.10	1.00	0.88	0.81
RTD Ranking					X

The development experiences of the personnel of the RTD System were shown in the following table.

Table 26. Development Experiences of RTD System Development Team

Position	Application Experience (year)
System Architect and Developer	10
Developer 1	10
Developer 2	5
Developer 3	6
Developer 4	12
Developer 5	2
Documentation Manager	8
Quality Assurance Manager	18

The average number of years of application experience was over 8, so the application experience ranking of the RTD team is ranked as Very High.

7.1.2.3.4 Platform Experience (PEXP)

Including PEXP is very important to the issue of productivity; it recognizes the importance of understanding the use of more powerful platforms including a larger graphical user interface (GUI) and database, networking, and distributed middleware experience [Boehm et al. 98].

Table 27. Platform Experience

	Very Low	Low	Nominal	High	Very High
PEXP	2 months	6 months	1 year	3 years	6 years
Value	1.19	1.09	1.00	0.91	0.85
RTD Ranking				X	

The personnel developing the RTD System had had more than 3 years, but less than 6 years of platform experience in Windows NT 4.0, Windows based GUI design, etc., so the platform experience of the RTD team was ranked as High.

7.1.2.3.5 Language and Tool Experience (LTEX)

This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem [Boehm et al. 98]. Software development includes the use of tools that perform requirements and design representation and analysis, configuration management, document extraction, library management, program style and formatting, consistency checking, etc. In addition

to experience in programming with a specific language, the supporting tool set also affects development time. A low rating is given for experience of less than 2 months with the language and the tools for the system being developed. A very high rating is given for a team averaging experience of 6 or more years with the language and tools.

Table 28. Language and Tool Experience

	Very Low	Low	Nominal	High	Very High
LTEX	2 months	6 months	1 year	3 years	6 years
Value	1.20	1.09	1.00	0.91	0.84
RTD Ranking				X	

The RTD System development company has had its own code standard, format standard, checking standard, etc. The developers were very familiar with the tools in developing, debugging, tracking, and version management. RTDI_A, the predecessor of the RTD system, was a DOS-based system. RTDI was being generated under Windows using a different set of programming languages and tools than those of RTDI_A. The language and tool experience of developers is more than 3 years but less than 6 years, so the ranking of the RTD team was High.

7.1.2.3.6 Personnel Continuity (PCON)

The rating scale for PCON is in terms of the project's annual personnel turnover, it ranges from 3% (very high) to 48% (very low) [Boehm et al. 98].

Table 29. Personnel Continuity

	Very Low	Low	Nominal	High	Very High
PCON	48% / year	24% / year	12% / year	6% / year	3% / year
Value	1.29	1.12	1.00	0.90	0.81
RTD Ranking					X

The developers of the RTD System had had similar project experiences and had more or less continually worked on this system and its predecessors, so the personnel continuity of the RTD team was assigned a ranking of Very High.

7.1.2.4 Project Factors

7.1.2.4.1 Use of Software Tools (TOOL)

The tool rating for tools used in the project/development ranges from simple edit and code (very low) to integrated life-cycle management tools (very high) [Boehm et al. 98].

Table 30. Use of Software Tools

	Very Low	Low	Nominal	High	Very High
TOOL	edit, code, debug	simple, front end, back end CASE, little integration	basic life-cycle tools, moderately integrated	strong, mature life-cycle tools, moderately integrated	strong, mature, pro active life-cycle tools, well integrated with processes, methods, reuse
Value	1.17	1.09	1.00	0.90	0.78
RTD Ranking				X	

The RTD System development team had an integrated programming environment, a software version control tool, an incident tracking database, etc. However, a project schedule and cost control management tool was not used, so the use of software tools for the RTD project is considered to be High.

7.1.2.4.2 Multisite Development (SITE)

Given the increasing frequency of projects being developed at a multitude of sites combined with the indications that multisite development effects are significant, the SITE cost driver has been added in cost estimation models [Boehm et al. 98]. Determining its cost driver rating involves the assessment and averaging of two factors: site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia).

Table 31. Multisite Development

	Very Low	Low	Nominal	High	Very High	Extra High
SITE: Communications	Some phone, mail	Individual phone, FAX	Narrow band email	Wideband electronic communication	Wideband electronic communication, occasional video conf.	Interactive multimedia
Value	1.22	1.12	1.00	0.90	0.81	
RTD Ranking	X					

Since the main development activities occur in a single site at the RTD development site, the RTD System SITE ranking was considered Very Low for this project.

7.1.2.4.3 Required Development Schedule (SCED)

This rating measures the schedule constraint imposed on the project team developing the software. The ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. Accelerated schedules tend to require more effort in the later phases of development because more issues are left to be determined due to lack of time to resolve them earlier [Boehm et al. 98]. A schedule compress of 75% is rated very low. A stretch-out of a schedule produces more effort in the earlier phases of development where there is more time for thorough planning, specification, and validation. A stretch-out of 160% is rated very high.

Table 32. Required Development Schedule

	Very Low	Low	Nominal	High	Very High
SCED	75% of nominal	85%	100%	130%	160%
Value	1.43	1.14	1.00	1.00	1.00
RTD Ranking				X	

A site visit and numerous communications with the developers of the RTD System informed us that the development schedule was usually within 150% of the

original schedule estimation. So the required development schedule of the RTD System was ranked as High.

The COCOMO intermediate model adjusts the calculation of effort by an Effort Adjustment Factor (EAF) [Boehm et al. 98]. The EAF is derived from 17 cost driver attributes. Based on the rating, the corresponding effort multiplier of each cost driver is used to compute the EAF. The EAF is the product of the rating weights of the 17 cost drivers in the four categories. The cost drives of the RTD System is shown in Table 33. The ratings were obtained / determined / calculated based on the information from Subsection 7.1.2.1 to Subsection 7.1.2.4.

Table 33. Ranking of the RTD System Cost Drivers

Cost drivers	Ranking	Value
Required Software Reliability (RELY)	Very high	1.26
Database Size (DATA)	Very high	1.28
Project Complexity (CPLX)	Nominal	1.00
Required Reusability (RUSE)	Very high	1.15
Documentation Match to Life-cycle Needs (DOCU)	Very high	1.23
Execution Time Constraint (TIME)	Extra high	1.63
Main Storage Constraint (STOR)	Nominal	1.00
Platform Volatility (PVOL)	Low	0.87
Analyst Capacity (ACAP)	Very high	0.71
Program Capacity (PCAP)	Very high	0.76
Application Experience (AEXP)	Very high	0.81
Platform Experience (PEXP)	High	0.91

Language and Tool Experience (LTEX)	High	0.91
Personal Continuity (PCON)	Very high	0.81
Use of Software Tools (TOOL)	High	0.91
Multisite Development (SITE)	Very low	1.43
Required Development Schedule (SCED)	High	0.93
Effort Adjustment Factor (EAF)		1.15

The Effort Adjustment Factor (EAF) of the RTD System was the product of the 17 cost driver values, which was 1.15.

7.1.3 Scaling Drivers

Equation Eq. 4.1-3 defines the exponent B. Table 34 below provides the rating levels for the COCOMO scale drivers. The selection of scale drivers was based on the rationale that they are a significant source of exponential variation on a project's effort or productivity variation. Each scale driver has a range of rating levels, from Very Low to Extra High. Each rating level has a weight, SF, and the specific value of the weight is called a scale factor. A project's scale factors, SFi, are summed across all of the factors, and used to determine a scale exponent, B, via Equation Eq 4.1-3.

For example, if scale factors with an Extra High rating are each assigned a weight of 0, then a 100 KSLOC project with Extra High ratings for all factors will have SFi = 0, B = 1.01, and a relative effort E = 105 PM. If scale factors with Very Low rating are each assigned a weight of 5, then a project with Very Low (5) ratings for all factors will have SFi= 25, B = 1.26, and a relative effort E = 331 PM. This represents a large variation, but the increase involved in a one-unit change in one of the factors is only about 4.7%.

Table 34. Scale Factors for the COCOMO Model [Boehm et al. 98].

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
FLEX	rigorous	occasional	relaxation	some relaxation	general conformity	general goals
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
PMAT	Weighted average of "Yes" answers to CMM Maturity Questionnaire					

7.1.3.1 Precedentedness (PREC) and Development Flexibility (FLEX)

These two scale factors largely capture the differences between the Organic, Semidetached, and Embedded modes of the original COCOMO model [Boehm et al. 98]. The following table maps project features onto the Precedentedness and Development Flexibility scales.

Table 35. Precedentedness and Development Flexibility

Feature	Very Low	Nominal / High	Extra High
Precedentedness			
Organizational understanding of product objectives	General	Considerable	Thorough
RTD Ranking			X

Experience in working with related software systems	Moderate	Considerable	Extensive
RTD Ranking			X
Concurrent development of associated new hardware and operational procedures	Extensive	Moderate	Some
RTD Ranking			X
Need for innovative data processing architectures and algorithms	Considerable	Some	Minimal
RTD Ranking		X	
Development Flexibility			
Need for software conformance with pre-established requirements	Full	Considerable	Basic
RTD Ranking	X		
Need for software conformance with external interface specifications	Full	Considerable	Basic
RTD Ranking	X		
Premium on early completion	High	Medium	Low
RTD Ranking		X	

For the RTD System, the Precedentedness was ranked as Extra High (based on the four factors in the table) and Development Flexibility was ranked as Low (based on the three factors in the table).

7.1.3.2 Architecture/Risk Resolution (RESL)

This factor combines two of the scale factors, "Design Thoroughness by Product Design Review (PDR)" and "Risk Elimination by PDR" [Boehm et al. 98]. The following table forms a comprehensive definition for the COCOMO rating levels. The RESL rating is the subjective weighted average of the listed characteristics.

Table 36. Architecture/Risk Resolution

Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Risk Management Plan identifies all critical risk items, establishes milestones for resolving them	None	Little	Some	Generally	Mostly	Fully
RTD Ranking				X		
Schedule, budget, and internal milestones Risk Management Plan	None	Little	Some	Generally	Mostly	Fully
RTD Ranking			X			
Percent of development schedule devoted to establishing architecture, given general product objectives	5	10	17	25	33	40
RTD Ranking				X		
Percent of required top software architects available to project	20	40	60	80	100	120
RTD Ranking		X				
Tool support available for resolving risk items, developing and verifying architectural specs	None	Little	Some	Good	Strong	Full
RTD Ranking		X				
Level of uncertainty in key architecture drivers: mission, user interface, COTS, hardware, technology, and performance	Extreme	Significant	Considera ble	Some	Little	Very Little

RTD Ranking				X		
Number and criticality of risk items	> 10 Critical	5-10 Critical	2-4 Critical	1 Critical	> 5 Non-Critical	< 5 Non-Critical
RTD Ranking			X			

Taking all of the areas into consideration and using weights for Very Low to Extra High (from 1 to 6), the average weight of the RTD System was (4 + 3 + 4 + 2 + 2 + 4 + 3)/7 = 3.1, near the value of Nominal. Thus the Architecture/Risk Resolution of the RTD System was considered Nominal.

7.1.3.3 Team Cohesion (TEAM)

The Team Cohesion scale factor accounts for the sources of project turbulence and entropy due to difficulties in synchronizing the project's stakeholders: users, customers, developers, maintainers, etc [Boehm et al. 98]. . These difficulties may arise from differences in stakeholder objectives and cultures, difficulties in reconciling objectives, and stakeholder's lack of experience and familiarity in operating as a team. The following table provides a detailed definition for the overall TEAM rating levels. The final rating is the subjective weighted average of the listed characteristics.

Table 37. Team Cohesion

Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Consistency of stakeholder objectives and cultures	None	Some	Basic	Considerable	Strong	Full
RTD Ranking						X

Ability, willingness of stakeholders to accommodate other stakeholders' objectives	None	Some	Basic	Consider able	Strong	Full
RTD Ranking						X
Experience of stakeholders in operating as a team	None	Little	Little	Basic	Consider able	Extens ive
RTD Ranking					X	
Stakeholder team building to achieve shared vision and commitments	None	Little	Little	Basic	Consider able	Extens ive
RTD Ranking						X

Taking all of the areas into consideration and using weights for Very Low to Extra High (from 1 to 6), the average weight for the RTD System was (6 + 6 + 5 + 6)/4 = 5.75, near the value of Extra High. Thus the Team Cohesion rating of the RTD development team was considered to be Extra High.

7.1.3.4 Process Maturity (PMAT)

The procedure for determining PMAT is organized around the Software Engineering Institute's Capability Maturity Model (CMM, which can be obtained from http://www.sei.cmu.edu/cmm). The rating of the Process Maturity is determined as the project starts.

The method is organized around the 18 Key Process Areas (KPA) in the SEI Capability Maturity Model [Paulk et al. 93]. The procedure for determining PMAT is to

decide the percentage of compliance for each of the KPA. If the project has undergone a recent CMM assessment, then the percentage compliance for the overall KPA (based on KPA Key Practice compliance assessment data) is used. If an assessment has not been done, then the levels of compliance to the KPA goals are used (with the scale below) to set the levels of compliance. The goal-based level of compliance is determined by a judgment based on averaging across the goals for each Key Process Area. The value for the RTD System is marked by an "X" in Table 38.

Table 38. Process Maturity

Key Process Areas	Almost Always (>90%)	Often (60-90%)	About Half (40-60%)	Occasion-ally (10-40%)	Rarely (<10%)	Does Not Apply	Don't Know
Requirements Management	X						
Software Project Planning	X						
Software Project Tracking and Oversight	X						
Software Subcontract Management						X	
Software Quality Assurance	X						
Software Configuration Management	X						
Organization Process Focus	X						
Organization Process Definition	X						
Training Program	X						
Integrated Software	X						

Management							
Software Product Engineering	X						
Intergroup Coordination	X						
Peer Reviews	X						
Quantitative Process Management		X					
Software Quality Management	X						
Defect Prevention		X					
Technology Change Management	X						
Process Change Management	X						

Note: How to mark the above table:

- Check "Almost Always" when the goals are consistently achieved and are well established in standard operating procedures (over 90% of the time).
- Check "Often" when the goals are achieved relatively often, but sometimes are omitted under difficult circumstances (about 60 to 90% of the time).
- Check "About Half" when the goals are achieved about half of the time (about 40 to 60% of the time).
- Check "Occasionally" when the goals are sometimes achieved, but less often (about 10 to 40% of the time).
- Check "Rarely" when the goals are rarely achieved (less than 10% of the time).

- Check "Does Not Apply" when you have the required knowledge about your project or organization and the KPA, but you feel the KPA does not apply to your circumstances.
- Check "Don't Know" when you are uncertain about how to respond for the KPA.

After the level of KPA compliance is determined, each compliance level is weighted and a PMAT factor is calculated as in Equation Eq. 7.1.3.4-1. Initially, all KPA will be equally weighted.

$$5-\left[\sum_{i=1}^{18}\left(\frac{KPA\%_i}{100}\times\frac{5}{18}\right)\right] \qquad \text{Eq 7.1.3.4-1}$$

For the RTD System, Process Maturity was ranked as Extra High.

The summary of the scale factors for the RTD System is shown in the Table 39 below.

Table 39. Summary of the RTD System Scale Factors

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC						X
FLEX		X				
RESL			X			
TEAM						X
PMAT						X

Now the scale exponent B was 0.91 + 0.01 * (5 + 1 + 2 + 5 + 5) = 1.09 (using Eq. 4.1-3). So the scale exponent of the RTD System was 1.09.

7.1.4 Lines of Code

The software tools used to determine the number of lines of code were CodeCount (which can be downloaded at the URL http://sunset.usc.edu/research/CODECOUNT/index.html) and Understanding for C++ developed by Scientific Toolworks, Inc. (which can be obtained at the URL http://www.scitools.com/downloadc.shtml). Some minor modifications (such as the TABs being replaced by blank line) were made to CodeCount to make it work for RTDI_A and the RTD System. The results of using these two tools to determine the number of lines of code of RTDI_A and the RTD System are given below.

- RTDI_A:
  - CodeCount: total lines 151,319, blank lines 19,696, comment lines 54,635, embedded comment lines 10,528, LOC 76,988 (this is the direct result of CodeCount, some lines of code are counted both in LOC and comment lines). Average percentage of comments to LOC for the RTDI_A software: (comment lines + embedded comment lines)/LOC = (54,635 + 10,528) / 76,998 = 84.6%.
  - Understanding for C++: total lines 136,681, blank lines 17,777, comment lines 59,899 (including comment lines and embedded comment lines), LOC 68,348. Average percentage of comments to LOC: comment lines /LOC = 59,899 / 68,348 = 87.0%.

- RTD System:
  - CodeCount: total lines 244,084, blank lines 29,178, comment lines 98,010, embedded comment lines 15,369, LOC 116,893. Average percentage of comments to

LOC: (comment lines + embedded comment lines)/LOC = (98,010 + 15,369) / 116,893 = 97.0%.

- Understanding for C++: total lines 254,274, blank lines 30,358, comment lines 120,535 (including comment lines and embedded comment lines), LOC 119,817. RTDI has 5,843 C++ functions and 320 C++ classes. Average percentage of comments to LOC: comment lines /LOC = 120,535 / 119,817 = 100.0%.

Compared to RTDI_A, RTDI (i.e., the RTD System) is not complete and not stable. This statement is based on the number of years RTDI_A has been in the field and the fact that RTDI was undergoing operational testing at the time of this research. Also, it must be mentioned that "not complete" does not mean "incomplete" and hence lacking or deficient; in fact, it can be argued that at any delivery point in time the software is complete with respect to the particular baselining/specification criteria utilized. Consequently, 120,000 was used as the most likely number of LOC for RTDI, taking +/- 10% for the sensitivity study.

For the RTD System, there is another component of the source code called RTDII, which is the communication service software. Since access could not be gained access to the source code and all relevant system documentation of RTDII, it was assumed that this software would be 10-15% of the RTDI source code (based on informal communications).

For sensitivity analysis, nine basic scenarios for LOC of the RTD System were studied. The LOC of the RTD System for the best scenarios is given below.

RTDI	Percentage of RTDII to RTDI	RTDII	LOC of the RTD System
120,000	12.5%	15,000	135,000

7.1.5 Estimation of RTD System

Based on the information obtained from Subsections 7.1.1 to 7.1.4 and applied to the model described in Chapter IV, development cost, phase distribution, and maintenance cost of the RTD System were obtained, as shown in Tables 40, 41, and 42.

Table 40. Development Cost of the RTD System

RTD System	Total LOC	Schedule (month)	Total Effort (person-month)	Total Cost ($)	Cost per Instruction	Staff Required
Optimistic		20.9	350.9	2,456,517	18.2	16.8
Most Likely	135,000	22.3	438.7	3,070,646	22.7	19.7
Pessimistic		23.8	548.3	3,838,308	28.4	23.0

Table 41. Phase Distribution of the RTD System

Overall	Percentage of Schedule	Schedule (month)	Percentage of Effort	Effort (person-month)	Staff Required
Plans And Requirements	0.18	4.91	0.07	30.71	6.26
Product Design	0.22	6.02	0.16	74.57	12.39
Programming	0.36	9.81	0.51	241.25	24.59
Integration and Test	0.24	6.47	0.26	122.84	18.99

EFFORT	Plans and Requirements	Product Design	Programming	Integration and Test
Requirements Analysis	13.82	9.32	9.65	3.07
Product Design	5.37	30.57	19.30	6.14
Programming	1.69	10.07	136.30	47.91
Test Planning	1.23	4.47	13.27	3.69
Verification and Validation	2.30	5.59	20.51	35.01
Project Office	3.84	7.46	14.47	8.60
CM/QA	0.92	1.86	15.68	9.83
Manuals	1.54	5.22	12.06	8.60

Personnel	Plans and Requirements	Product Design	Programming	Integration and Test
Requirements Analysis	2.82	1.55	0.98	0.47
Product Design	1.10	5.08	1.97	0.95
Programming	0.34	1.67	13.89	7.41
Test Planning	0.25	0.74	1.35	0.57
Verification and Validation	0.47	0.93	2.09	5.41
Project Office	0.78	1.24	1.48	1.33
CM/QA	0.19	0.31	1.60	1.52

Manuals	0.31	0.87	1.23	1.33

Table 42. Estimated Maintenance Cost of the RTD System

Life-cycle Year	Maintenance Cost ($)
1	84,444
2	85,283
3	86,129
4	42,430
5	42,430
6	42,430
7	21,320
8	21,320
9	21,320
10	21,320
11	21,320
12	21,320
13	21,320
14	21,320
15	21,320
16	21,320
17	21,320
18	21,320
19	21,320
20	21,320

Total	681,629

So the total life-cycle cost of the RTD System was estimated to be $3,752,275.

7.1.6 Risk Assessment

7.1.6.1 Using Software Risk Assessment Model

The questionnaire related to the SRAM model and the responses for the RTD System are shown in Appendix B. For all questions with respect to each particular risk element, the choices were obtained and their numerical ratings were accumulated. For the RTD System, values of the responses obtained are summarized in the following table. The rows are a sequence of questions and the columns are software risk question types.

Table 43. Values of the RTD System Software Risk Questions

	Complexity	Staff	Reliability	Requirement	Estimation	Monitoring	Development	Usability	Tools
1	1	1	2	1	1	1	1	1	1
2	1	1	2	1	2	3	1	1	3
3	1	1	1.7	2	1	3	1	1	13
4	3	2	2	1	1	3	1	1	1
5	3	1	2	2	2	3	1	1	3
6	3	1	1	3	1	1	1	1	3
7	1	1	1.7	1	1	1	1		1
8	2	1	1	1	2	1	1		1
9	1	1	1	1		2	1		
10	2	1	1	1			1		
11		1	2				1		

12			1				1		
13							1		
total	18	12	18.4	14	11	18	13	6	26

The quality risk of the RTD System can be calculated in three different ways as follows:

$R_{quality\text{-}max} = 60;$
$R_{quality\text{-}min} = 20;$
$R_{quality} = 27.97;$
$R_{quality\text{-}n} = (27.97 - 20)/(60 - 20) = 0.20$

$R_{quality\text{-}max} = 54;$
$R_{quality\text{-}min} = 18;$
$R_{quality} = 26.04;$
$R_{quality\text{-}n} = (26.04 - 18)/(54 - 18) = 0.22$

$R_{quality\text{-}max} = 51;$
$R_{quality\text{-}min} = 17;$
$R_{quality} = 24.04;$
$R_{quality\text{-}n} = (24.04 - 17)/(51 - 17) = 0.19$

Thus the overall risk of the RTD System was $(0.20 + 0.22 + 0.19)/3=0.20$.

The level of risk in relation to schedule and cost can be obtained in a similar fashion.

If the risk level of a project assessed by SRAM is above 0.6, this indicates that it is a high-risk project. In such a case, it is strongly advised to discontinue or not to undertake the project [Foo and Muruganantham 00].

If the risk level of a project assessed by SRAM is between 0.36 and 0.6, it is necessary to reduce the likelihood and impact of the risk elements through immediate risk containment procedures. The project manager should identify the weak areas that require urgent attention and resources.

If the risk level of a project assessed by SRAM is below 0.36, the project is a low risk project and the chances of success are high.

For the RTD System, the quality risk level was assessed to be low, the schedule risk level was assessed to be low, and the cost risk level was assessed to be low. Thus the overall risk level of the RTD System was assessed to be low.

7.1.6.2 Cost Factors Heuristic Risk Assessment Model

For the RTD System, using the cost drivers in Subsection 7.1.2 and the scale drives in Subsection 7.1.3, after calculation, the results are shown in the Table 44.

Table 44. Values of the RTD System Software Risk

	Schedule risk	Product risk	Platform risk	Personnel risk	Process risk	Reuse risk	Overall risk
Risk Value	3.56	1.98	3.51	0	0	0	2.43
Risk Level	Low	Low	Low	Low	Low	Low	Low

We can see that all risk levels of the RTD System are low, which is essentially the same as the risk assessment result we arrived at using SRAM in the previous section.

7.2 Operating System Simulation (OSS)

This simulation project includes two parts: phase one whose file was called p1.cpp and phase two whose file was named p2.cpp.

7.2.1 LOC

The result of using CodeCount (which can be downloaded at the URL http://sunset.usc.edu/research/CODECOUNT/index.html) on p1.cpp are listed below.

Total lines 1,305, blank lines 196, whole comment lines 315, embedded comment lines 136, LOC 794 (this is the direct result of CodeCount, some lines of code were counted both in LOC and comment lines). The average percentage of comments to LOC: (comment lines + embedded comment lines)/LOC = 315 / 794 = 40.0%.

The results of using CodeCount on p2.cpp are listed below.

Total lines 2,594, blank lines 238, whole comment lines 410, embedded comment lines 248, LOC 1, 946 (this is the direct result of CodeCount, some lines of code were counted both in LOC and comment lines). The average percentage of comments to LOC: (comment lines + embedded comment lines)/LOC = 410 / 1946 = 21.1%.

So the final results for the simulation project were:

Total lines 3,899, blank lines 434, whole comment lines 725, embedded comment lines 384, LOC 2,740 (this is the direct result of CodeCount, some lines of code are counted both in LOC and comment lines). The average percentage of comments to LOC: (comment lines + embedded comment lines)/LOC = 725 / 2740 = 26.5%.

7.2.2 Cost and Schedule Estimation

The Effort Adjustment Factor (EAF) of the simulation project was calculated and is shown in Table 45 below.

Table 45. EAF of the Simulation Project

Cost drivers	Ranking	Value
Required Software Reliability (RELY)	High	1.10
Database Size (DATA)	Low	0.90
Project Complexity (CPLX)	High	1.10
Required Reusability (RUSE)	Nominal	1.00

Documentation Match to Life-cycle Needs (DOCU)	Nominal	1.00
Execution Time Constraint (TIME)	Very high	1.29
Main Storage Constraint (STOR)	Nominal	1.00
Platform Volatility (PVOL)	Low	0.87
Analyst Capacity (ACAP)	Very high	0.71
Program Capacity (PCAP)	Very high	0.76
Application Experience (AEXP)	High	0.88
Platform Experience (PEXP)	High	0.91
Language and Tool Experience (LTEX)	High	0.91
Personal Continuity (PCON)	Very high	0.81
Use of Software Tools (TOOL)	Nominal	1.00
Multisite Development (SITE)	Very low	1.43
Required Development Schedule (SCED)	High	0.93
Effort Adjustment Factor (EAF)		0.52

The Effort Adjustment Factor (EAF) of the simulation project is the product of the 17 cost driver values, which was 0.52.

The scale factors for the simulation project were also calculated and are shown in Table 46 below.

Table 46. Scale Factors of the Simulation Project

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC				X		

FLEX					X	
RESL			X			
TEAM						X
PMAT				X		

The scale exponent B was 0.91 + 0.01 * (3 + 4 + 2 + 5 + 3) = 1.16 (using Equation Eq. 4.1-3). So the scale exponent of the simulation project was 1.16.

The development cost of the Simulation Project is shown in Table 47.

Table 47. Development Cost of the Simulation Project

Simulation Project	Total LOC	Schedule (month)	Total Effort (person-month)	Total Cost ($)	Cost per Instruction	Staff Required
Optimistic		3.3	3.90	16,583	6.1	0.6
Most Likely	2,740	4.1	4.43	20,729	7.6	0.7
Pessimistic		5.2	5.95	25,911	9.5	0.9

The phase distribution of the Simulation Project is shown in Table 48.

Table 48. Phase Distribution of the Simulation Project

Overall	Percentage of Schedule	Schedule (month)	Percentage of Effort	Effort (person-month)	Staff Required
Plans And Requirements	0.14	0.92	0.07	0.29	0.32
Product Design	0.21	1.36	0.16	0.70	0.52

Programming	0.48	3.13	0.60	2.64	0.84
Integration and Test	0.17	1.15	0.17	0.80	0.70

EFFORT	Plans and Requirements	Product Design	Programming	Integration and Test
Requirements Analysis	0.14	0.09	0.11	0.02
Product Design	0.05	0.29	0.21	0.04
Programming	0.01	0.09	1.49	0.27
Test Planning	0.01	0.03	0.11	0.02
Verification and Validation	0.02	0.04	0.19	0.26
Project Office	0.04	0.09	0.20	0.07
CM/QA	0.01	0.02	0.18	0.07
Manuals	0.02	0.06	0.16	0.06

Personnel	Plans and Requirements	Product Design	Programming	Integration and Test
Requirements Analysis	0.15	0.06	0.03	0.02
Product Design	0.05	0.21	0.07	0.03
Programming	0.01	0.06	0.48	0.23
Test Planning	0.01	0.02	0.03	0.02
Verification and	0.02	0.03	0.06	0.22

Personnel	Plans and Requirements	Product Design	Programming	Integration and Test
Validation				
Project Office	0.05	0.07	0.06	0.06
CM/QA	0.01	0.02	0.06	0.06
Manuals	0.02	0.04	0.05	0.06

The estimated maintenance cost of the Simulation Project in a 20 years life-cycle is shown in Table 49.

Table 49. Estimated Maintenance Cost of the Simulation Project

Life Cycle Year	Maintenance Cost ($)
1	9,544
2	9,739
3	9,937
4	5,672
5	5,788
6	5,906
7	1,852
8	1,852
9	1,852
10	1,852
11	1,852
12	1,852
13	1,852

14	1,852
15	1,852
16	1,852
17	1,852
18	1,852
19	1,852
20	1,852
Total	72,530

So the total life-cycle cost of the simulation project was $93,259.

7.2.3 Complexity

Using PCMetric 4.0 DOS-based software (which was developed by Set Laboratories, Inc. and can be purchased at http://www.molalla.net/~setlabs) to obtain the complexity of the simulation project, the following quantitative results were obtained.

- Unique Operators (n1) 138, Unique Operands (n2) 318, Total Operators (N1) 8,226, and Total Operands (N2) 5,143.
- Software Science Length (N) 13,369, Software Science Volume (V) 118,087, and Software Science Effort (E) 131,777,190.
- Cyclomatic Complexity (VG1) 370, Extended Cyclomatic Complexity (VG2) 586, Average Cyclomatic Complexity 2, and Average Extended Cyclomatic Complexity 4.

7.2.4 Maintainability

Using the three metric Maintainability Index (MI) equations discussed in Chapter V, we get the maintainability index of the Simulation Project.

$MI = 171 - 5.2ln(Avg\ V) - 0.23AvgV(g) - 17.2ln(Avg\ LOC) = 171 - 5.2ln(118,087/(47+84)) - 0.23ln2 - 17.2ln(2740/(47+84)) = 83.2$

According to quality cutoffs for analyzing systems with polynomial metrics [Coleman 92], 83.2 is between 65 and 85, so the maintainability of the simulation project was classified as moderate.

7.2.5 Risk Assessment

Using the Cyclomatic Complexity Risk Assessment Model discussed in Subsection 6.3, the average Cyclomatic complexity of the simulation project was 2, which is between 1 and 10, so we can say that the risk level of the simulation project is low.

Chapter VIII

CONCLUSION AND FUTURE WORK

Software maintainability is a complex subject. The main objectives of this thesis was to investigate and build a comprehensive but easy to follow framework for a software maintainability study. By using the methods discussed in this thesis, a software maintainability study can be carried out reasonably painlessly.

A software maintainability study contains four steps. The first step is to use lines of code, function points, or object points to size the software. The second step is to use COCOMO II or COSMOS to calculate the software development and maintenance effort. The third step is to use McCabe's Cyclomatic and Halstead's measures to calculate a software maintainability index. The last step is to analyze software risk using SRAM, CFHRAM, or CCRAM based on the characteristics of the software. The two case studies in Chapter VII showed that the framework has sufficient flexibility and can be followed effectively and easily.

LOC and FP should be used to calibrate for each other. COCOMO and COSMOS can be chosen to perform software cost and schedule estimation. The three risk analysis methods, SRAM, CFHRAM and CCRAM, should be chosen carefully based on relevance and applicability.

Future work should concentrate on the following areas.

- Building a comprehensive set of tools which would combine all methods discussed in this thesis.
- Tools should be able to point out the weak area of the software and suggest some improvement techniques.
- Function Points should be automatically counted using built-in software components.
- The SRAM model should consist of several different levels of questionnaires according to the complexity of the software under consideration.
- Objected Oriented program measures should be used.

REFRENCES

[Albrecht and Gaffney 83] Allan J. Albrecht and John E. Gaffney, Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 6, pp. 639-648, November 1983.

[Boehm 81] Barry W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Boehm 96] Barry W. Boehm, "Anchoring the Software Process", *IEEE Software*, Vol. 13, No. 4, pp. 73-82, July 1996.

[Boehm et al. 98] Barry W. Boehm, Christ Abts, Brad Clark, and Sunita Devnani-Chulani, "COCOMO II Model Manual", *COCOMO II Research Project*, http://sunset.usc.edu/research/ COCOMOII/, date accessed: June 2, 2001, date created: October 2, 1998.

[Coleman 92] D. Coleman, "Assessing Maintainability," *Proceedings of the 1992 Software Engineering Productivity Conference,* pp. 525-532, San Jose, CA, May 1992.

[Dale 96] Walker K. Dale, *Software Engineering Risk Management*, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[Foo and Muruganantham 00] Say-Wei Foo and A. Muruganantham, "Software Risk Assessment Model", *Proceedings of the 2000 IEEE International Conference on Management of Innovation and Technology*, pp. 536-544, Singapore, November 2000.

[Halstead 77] Maurice H. Halstead, *Elements of Software Science,* Elsevier, New York, NY, 1977.

[Henry 98] Joel Henry, "The Software Cost Modeling System (COSMOS)", *COSMOS Research Project*, http://www-cs.etsu-tn.edu/cosmos/, date accessed: June 6, 2001, date created: April 21, 1998.

[Ikatura and Takayanagi 82] Minoru Ikatura and Akio Takayanagi, "A Model for Estimating Program Size and Its Evaluation", *Proceedings of the Six International Conference on Software Engineering*, pp. 104-109, Tokyo, Japan, September 1982.

[Jones 91] Capers Jones, *Applied Software Measurement*, McGraw-Hill, New York, NY, 1991.

[Jones 99] Capers Jones, "Software Sizing", *IEE Review*, Vol. 45, No. 4, pp. 165-167, July 1999.

[Madachy 97] R. J. Madachy, "Heuristic Risk Assessment Using Cost Factors", *IEEE Software*, Vol. 14, No. 3, pp. 51-59, May/June 1997.

[McCabe and Watson 94] Thomas J. McCabe and Arthur H Watson, "Software Complexity", *Journal of Defense Software Engineering*, Vol. 7, No. 12, pp. 5-9, December 1994.

[Najberg 84] Andrew C. Najberg, "Software Data Base Development", *The Analytical Science Corporation Technical Report 4612-S-1*, Vol. I, pp. 2-4, Burlington, MA, March 1984.

[Oman and Hagemeister 92] P. Oman and J. Hagemeister, "Construction and Validation of Polynomials for Predicting Software Maintainability", *Software Engineering Test Lab Report #92-01 TR*, Computer Science Department, University of Idaho, Moscow, ID June 1992.

[Oman and Hagemeister 94] P. Oman and J. Hagemeister, "Constructing and Testing of Polynomials for Predicting Software Maintainability", *Journal of Systems and Software,* Vol. 24, No. 3, pp. 251-266, March 1994.

[Park 92] Robert E. Park, "Software Size Measurement: A Framework for Counting Source Statements", *Technical Report CMU/SEI-92-TR-20*, Pittsburgh, PA, September 1992.

[Paulk et al. 93] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model, Version 1.1," *IEEE Software*, Vol. 10, No. 4, pp. 18-27, July 1993.

[Peters and Pedrycz 00] James F. Petes and Witold Pedrycz, *Software Engineering: An Engineering Approach*, John Wiley & Sons, Inc., New York, NY, 2000.

[Pressman 01] Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, 5th ed., McGraw-Hill, New York, NY, 2001.

[Putman 80] L. Putnam, *Software Cost Estimation and Life Cycle Control*, IEEE Computer Society, Los Alamitos, CA, 1980.

[RCA PRICE Systems 87] RCA PRICE Systems, *PRICE-SZ Reference Manual*, Moorestown, MT, 1987.

[Reifer 86] Donald J. Reifer, *A Poor Man's Guide to Estimating Software Costs, Tutorial Software Management, 3rd* ed. IEEE Computer Society Press, Washington DC, 1986.

[Zhou et al.93] F. Zhou, B. Lowther, P. Oman, and J. Hagemeister, "Constructing and Testing Software Maintainability Assessment Models", *Proceedings of the First International Software Metrics Symposium*, pp. 61-70, Baltimore, MD, May 1993.

APPENDICES

APPENDIX A

GLOSSARY

CC	Cyclomatic Complexity (a quantitative measure of the logical complexity of a program, the value defines the number of independent paths in a program).
CCRAM	Cyclomatic Complexity Risk Assessment Model.
CD	Cost Driver (factors that influence software cost estimation, e.g., developer's experience, platform factor, and time factor).
CFHRAM	Cost Factor Heuristic Risk Assessment Model.
COCOMO	COnstructive COst MOdel.
COSMOS	COSt MOdeling System.
DET	Data Element Type (distinct fields contained within the logical groups of data).
EI	External Input (data or control information that comes into the application to perform an elementary process).
EIF	External Interface File (logical data groups that provide information to an application but are maintained by another application).
EO	External Output (calculated or derived data that exit an application).
EQ	External Inquiry (input/output combination that supplies a body of data in response to a request from outside the application).
FP	Function Point (Function Points measures the functionality visible to and delivered to the user. Functionality is divided into two kinds: Data Functionality supplied through logical groups of data that are read or

	maintained and Transaction Functionality supplied through processes provided by the software).
FTR	File Type Referenced (number of ILFs and EIFs used by a program).
IFPUG	International Function Point Users Group.
ILF	Internal Logical File (logical data groups that are maintained within an application boundary).
JOP	Adjusted Object Points (the number of object points after considering the influences of reuse factors).
LOC	Lines of Code.
MI	Maintainability Index (a measure that attempts to quantify the maintainability of software).
OP	Object Point (an indirect software measure that is computed using the counts of number of screens, reports, and components likely to be required to build an application).
PERT	Program Evaluation and Review Technique equation (used to calculate the size of software in Subsection 3.3).
PM	Person Month of estimated effort.
RET	Record Element Type (distinct record formats of the data logical group used in identifying function points).
SCED	Schedule.
SEI	The Software Engineering Institute in Carnegie Mellon University, Pittsburgh, PA.
SERIM	Software Engineering Risk Model.
SLOC	Source Lines of Code (interchangeable with the lines of code or LOC).
SRAM	Software Risk Assessment Model.
UFP	Unadjusted Function Point Count which is the simple count of function points without being applied to the conversion factors.

APPENDIX B

QUESTIONNAIRE RELATED TO THE SRAM MODEL

What follows is the questionnaire related to the SRAM model and the responses for the RTD System (the real time display system).

	RTD
I. Complexity of Software	
Q.1 What is the function of the software to be developed?	
a. Data processing software	X
b. Service software (Communication software)	
c. System software	
Q.2 What is memory limitation?	
a. Allowed the use of any amount of memory	X
b. Some additional memory allowed	
c. Restricted to a specific amount of memory	
Q.3 What is the complexity of I/O device?	
a. Standard I/O device	X
b. Medium complexity I/O devices (e.g., scanner)	
c. High complexity I/O devices (e.g., microphone)	
Q.4 What are the time requirements/limits for the functions?	
a. No function requires time limits	
b. Certain function requires time limits	

	RTD
c. Many functions require time limits	X
Q.5 What is the hardware (or platform) dependency?	
a. Platform independent (e.g., Java programs)	
b. Run on some platforms only (e.g., GNU C programs)	
c. Highly platform independent (e.g., Tandem C programs)	X
Q.6 What type of control operations is used in the programs?	
a. Straight line cede, few nested structured programming operations	
b. Simple nesting, some inter-module control, simple message passing	
c. Multiple resource scheduling, dynamically changing priorities	X
Q.7 What type of computational operations is used in the programs?	
a. Evaluation of simple expressions	X
b. Use of standard mathematics and statistical routines, basic matrix/vectors	
c. Difficult and unstructured numerical analysis/ stochastic data	
Q.8 What type of device dependent operations is used in the programs?	
a. Simple read/write statements with simple formats	
b. I/O procession includes device selection/status checking	X
c. Device time dependent / micro programmed operations	
Q.9 What type of data management operations is used in the programs?	
a. Simple array in main memory, simple database queries, updates	X
b. Multiple input, single output, simple structural changes	
c. Highly coupled, dynamic relational and object structures	
Q.10 What type of user interface management operations is used in the programs?	
a. Simple input forms/reports generations	
b. Use of widget sets	X

	RTD
c. Complex multimedia, virtual reality	
II. Staff Involved in the Project	
Q.1 What is the average experience of staff members with the company?	
a. More then 5 years	X
b. About 3 years	
c. Less than a year	
Q.2 What is the average coding experience of a staff member?	
a. Exceptional	X
b. Average, at least one expert	
c. Beginners	
Q.3 What is the motivation level of staff?	
a. High, enjoys working, no complaints	X
b. Moderate, some complaints	
c. Low, a lot of complaints	
Q.4 What is the level of knowledge of staff in the application domain?	
a. Have worked on the whole process several times	
b. Have worked on several portions of the system	X
c. Have reading knowledge of the application	
Q.5 What is the average number of lines of code (C equivalent) produced per person per day?	
a. 30 lines	X
b. 50 lines	
c. More than 70 lines	
Q.6 What is the variety or mix of software disciplines/ experiences on the team?	
a. Good mix of all software disciplines	X

	RTD
b. Some disciplines are inappropriately presented	
c. Some disciplines are not present	
Q.7 What is the experiences level of the software manager(s)?	
a. Have had similar management experiences.	X
b. Mix of new manager(s) and experienced manager(s)	
c. New manager(s) with software engineering knowledge	
Q.8 Does each member of the staff have a career plan with goal setting?	
a. All members have a goal and career plan reviewed regularly	X
b. Some members may have career plan, but not followed closely	
c. No career plan and goal setting	
Q.9 What is the level of harmony among the staff?	
a. Harmonious with good teamwork	X
b. Some unhealthy arguments at time	
c. No compromises even in the fundamentals	
Q.10 Does every employee clearly understand his/her roles and responsibilities?	
a. Clearly understands the roles and responsibilities	X
b. Generally understands the roles and responsibilities	
c. Confuses about the roles and responsibilities	
Q.11 Is there a proper reward mechanism?	
a. A good performance appraisal and reward scheme is in place	X
b. *Ad hoc* performance appraisal and reward system	
c. No performance appraisal and reward system	
III. Targeted Reliability	
Q.1 Are there error handling conditions throughout the program?	

	RTD
a. Error handling conditions for every possible instances	
b. Error handling conditions for some possible instances	X
c. No Error handling conditions within the programs	

Q.2 How are error conditions handled?

a. Processing continues for any error condition	
b. Processing continues for some error conditions	X
c. Processing discontinues upon any error condition	

Q.3 Are error tolerance conditions defined for input and output?

a. All error tolerance conditions are defined	X 30%
b. Some error tolerance conditions are defined	X 70%
c. No error tolerance conditions are defined	

Q.4 Are input checked for validity before processing?

a. All inputs are checked	
b. Some inputs are checked	X
c. No inputs are checked	

Q.5 Are hardware faults detected and processed in the software?

a. All hardware faults are detected and processed	
b. Some hardware faults are detected and processed	X
c. No hardware faults are detected and processed	

Q.6 Is the use of global data types minimized in the software?

a. Few and no global data type are used	X
b. Some global data types are used	
c. Global data types are heavily used	

Q.7 Are defect data collected during software integration?

a. All defect data are collected	X 30%
b. Some defect data are collected	X 70%

	RTD
c. No defect data are collected	
Q.8 Are defect data logged-in and closed-out prior so the delivery?	
a. All defect data is logged in and closed out	X
b. Some defect data are logged in and closed out	
c. No defect data is logged in and closed out	
Q.9 Are all the requirements tested?	
a. All the requirements are tested	X
b. Some of the requirements are tested	
c. No requirements are tested	
Q.10 Is stress testing performed (for the changes in codes)?	
a. Stress testing is performed on all software	X
b. Stress testing is performed for some software	
c. No requirements are testing	
Q.11 Who performs the system testing?	
a. Independent test team(s)	
b. An Independent test team(s) and developers share the testing	X
c. Developer(s) do all the testing themselves	
Q.12 Does the company have similar past experience(s) in developing this type of software?	
a. Has track record producing similar products	X
b. Some experiences in similar products	
c. No past experiences	
IV. Product Requirements	
Q.1 Are all the software requirements identified and/or documented?	
a. All requirements are identified and documented	X

	RTD
b. Some requirements are identified and documented	
c. No requirements are identified and documented	
Q.2 Is customer involved in the definition of requirements?	
a. Customer is heavily involved	X
b. Customer is partially involved	
c. Customer in not involved	
Q.3 Does the customer approve all requirements?	
a. All requirements are approved by the customer	
b. Some of the requirements need customer approval	X
c. The customer does not approve all requirements	
Q.4 Are ambiguous requirements verified through prototyping?	
a. Ambiguous requirements are verified through prototyping	X
b. Some ambiguous requirements are verified through prototyping	
c. Ambiguous requirements are not verified through prototyping	
Q.5 Are requirement categorized as essential, nice-to-have etc.?	
a. All requirement are categorized and prioritized	
b. Some requirement are categorized and/or prioritized	X
c. Requirement are neither categorized nor prioritized	
Q.6 Are there any differences between customer requirements and software requirements (used by the developers)?	
a. No differences, i.e. customer requirement are intact	
b. Some refinement on requirements	
c. Many refinement on requirements	X
Q.7 Are software requirements frozen before the subsequent phase?	
a. All requirements are frozen before the next phase	X
b. Some requirement are expected to be changed	
c. No requirement are expected to be changed	

RTD

Q.8 Are software requirements traceable to code?

a. All software requirements are traceable to code X

b. Some software requirements are traceable to code

c. No software requirements are traceable to code

Q.9 Are software requirements traceable to test procedures?

a. All software requirements are traceable to test procedures X

b. Some software requirements are traceable to test procedures

c. No software requirements are traceable to test procedures

Q.10 Are all the open action items closed prior to delivery to the customer?

a. All the open actions Items are addressed and implemented X

b. Some open actions Items are addressed and implemented

c. No action items are addressed or implemented

V. Method of Estimation

Q.1 What Is the estimation method used?

a. Bottom-up X

b. Analogy, Top-down

c. Other techniques

Q.2 Is there any model used to compute the cost of the project?

a. A suitable cost model is used

b. A model is used for partial cost estimation X

c. No cost model is used

Q.3 Is estimation based on past software productivity metrics?

a. Based on the past software productivity metrics (similar projects) X

b. Partially based on the past software productivity metrics

c. Not based on the past software productivity metrics

	RTD
Q.4 Are schedule estimates based on past software projects?	
a. Based on similar software project metrics	X
b. Partially based on similar software project metrics	
c. Not based on similar software project metrics	
Q.5 How often are estimates revised?	
a. Estimates are updated on monthly or on more frequent basis	
b. Estimates are updated at the end of the phases	X
c. Estimated are never updated	
Q.6 How accurate are the past schedule estimates compared to actual schedule?	
a. Varies within (+/-)5% range of actual schedule	
b. Varies within (+/-)50% range of the actual schedule	X
c. Varies more than (+/-)100% of the actual schedule	
Q.7 What is the level of participation of developers in the estimation?	
a. All developers participated	X
b. Some developers participated	
c. Only manager prepare the estimate	
Q.8 What are the resources allocated for the estimation?	
a. All required resources are allocated	
b. Some resources are allocated	X
c. No resources are allocated	

VI. Method of Monitoring

Q.1 Are there distinct milestones for each major software effort?	
a. Distinct milestones for each development phase	X

	RTD
b. Some milestones	
c. No milestones	
Q.2 Is a detailed Work Breakdown Structure (WBS) used to track mid report cost and budget for each phase of the software development?	
a. Cost structure or WBS exists and properly tracked and reported	
b. Cost structure or WBS exists but not properly tracked and reported	
c. No cost Structure or WBS	X
Q.3 Is there a monitoring system?	
a. A monitoring system tracks cost, schedule and earned value	
b. A monitoring system exists but does not tracks cost, schedule and earned value	
c. No monitoring system is used	X
Q.4 How often are the project progress reports created?	
a. Weekly	
b. Monthly	
c. No project report created	X
Q.5 How often are cost, schedule and earned value reports updated?	
a. Monthly or frequently	
b. Updated less regularly (e.g., quarterly or less frequently)	
c. Never updated	X
Q.6 How frequent is the problem/action log updated?	
a. Weekly	X
b. Updated less frequently than a week	
c. Problem log or action log system does not exist	
Q.7 How often are the records for technical problems updated?	
a. Weekly or more frequently	X
b. Longer than a weekly basis	

	RTD
c. No records are kept	
Q.8 How is the schedule plan developed?	
a. Bottom up, all members are involved	X
6. Bottom up, some Important members are not Involved	
c. Top down, developed by one manager	
Q.9 What is the span of control?	
a. Each superior supervises not more than three subordinates	
b. Superior supervises between four and six subordinates	X
c. Superior supervises more than six subordinates	
VII. Development Process Adopted	
Q.1 Is a software management planning document used for the project?	
a. Used and adhered closely	X
b. Used but not followed closely	
c. Not used	
Q.2 Are software configuration management functions performed?	
a. All software configuration management functions are performed	X
b. Some software configuration management functions are performed	
c. No software configuration management functions are performed	
Q.3 Does communication exist between different organizations supporting the development of the software project?	
a. Very good communication	X
b. Reasonable communication	
c. Poor communication	
Q.4 Are software developers trained in the development methodology?	
a. All are trained	X

	RTD
b. Some are trained/ all trained in some portion of methodology	
c. None is trained in the development methodology	
Q.5 How closely is the software development methodology followed?	
a. It is closely followed by all	X
b. It is followed by some.	
c. it is not followed	
Q.6 Are software quality functions performed?	
a. All of the software quality functions are performed.	X
b. Some software configuration management functions are performed	
c. No software quality functions are performed	
Q.7 Is the current development methodology suitable for the project?	
a. It is tailored to the process	X
b. It is a fixed process	
c. No process is followed	
Q.8 Does the development methodology addresses requirements, design, code views / walk through / inspections?	
a. Yes, addresses all	X
b. Not all of the above	
c. Does not address any of the above	
Q.9 Does the development methodology require test plans and / or test procedures for all software functions?	
a. It requires test plan and/or test procedures for all software functions	X
b. It requires test plan and/or test procedures for some software functions	
c. It does not require test plans and/or test procedures	
Q.10 Does the development methodology require documentation?	

	RTD
a. Development methodology requires documentation	X
b. Development methodology requires partial documentation	
c. Development methodology does not require documentation	

Q.11 Is regression testing performed?

a. Regression testing is performed for all subsystems	X
b. Regression testing is performed for some subsystems	
c. No regression testing is performed	

Q.12 Is there a documented organizational structure in place?

a. Documented organizational structure and operations are in place	X
b. No documented organizational structure but a clear line of authority is in place	
c. There is no documented organizational structure	

Q.13 Is the organizational structure stable?

a. No changes in the organizational structure	X
b. Some organizational changes but not frequent	
c. Frequent changes	

VIII. Usability of Software

Q.1 Will a user manual be written for the software product?

a. User manual will be developed, tested, and delivered with the product	X
b. User manual will not be verified against the software functions	
c. No user manual will be provided for the software product	

Q.2 Are there help functions for input or output screens?

a. Help functions are provided for each input or output function	X
b. Some help functions are provided for input or output functions	
c. No help functions are provided	

	RTD
Q.3 Is the user involved in reviewing prototypes or earlier version of the software?	
a. User is involved and feedback is solicited from the user	X
b. Some feedback is solicited from the user	
c. User is not involved	
Q.4 Is the user interface designed to industry standards or to standards familiar to the user?	
a. Industry standards or standards familiar to the user are followed	X
b. Some aspects of standards are followed	
c. No standards are followed	
Q.5 Are user response times identified?	
a. All user response times we identified	X
b. Some user response times are identified	
c. No user response times are identified	
Q.6 Is the design evaluated to minimize keystrokes and data entry?	
a. Entire design is evaluated to minimize keystrokes and data entry	X
b. Some considerations are made to minimize keystrokes and data entry	
c. No design consideration is made to minimize keystrokes and data entry	
IX. Tools Used for Development	
Q.1 Are software developers trained to use tools for development?	
a. All are trained to use tools	X
b. Some are trained to use tools	
c. No one is trained to use tools or no tool is available	
Q.2 Are automated software tools used for testing?	
a. Automated tools are used in testing and they are adequate	X

	RTD
b. Automated tools are used in testing but they we not adequate	
c. No automated tools are used in testing	
Q.3 Are automated tools used for code generation (i.e. screen painters)?	
a. Adequate automated tools are used for code generation	X
b. Some automated tools are used but nut adequate	
c. No automated tools are used	
Q.4 Are automated tools used for test procedure(s)?	
a. Adequate automated tools are used	
b. Same automated tools me used but not adequate	
c. No automated tools are used	X
Q.5 Are tools used for configuration management functions?	
a. Adequate configuration management tools are used	X
b. Some configuration management tools are used	
c. No configuration management tools are used	
Q.6 Are automated tools used for regression testing?	
a. Adequate automated tools are used	
b. Some automated tools are used, but not adequate	
c. No automated tools are used	X
Q.7 Are automated tools used for reengineering?	
a. Adequate automated tools are used	
b. Some automated tools are used, but not adequate	
c. No automated tools are used	X
Q.8 How stable is the compiler / linker / debugger?	
a. Stable with few or no problems	X
b. Somewhat stable, but with some known problems	
c. Not stable, with a lot of problems	

RTD

Q.9 Are required tools readily available to developers when needed?

a. All tools me available X

b. Some tools are available

c. No tools are available

2.

VITA

QI SUN

Candidate for the Degree of

Master of Science

Thesis: SOFTWARE MAINTAINABILITY STUDY: A FRAMEWORK

Major Field: Computer Science

Biographical:

Personal Data: Born in Shangdong, China, January 22, 1972, son of Mr. Yuzhong Sun and Mrs. Xiuqin W. Sun.

Education: Received Bachelor of Science degree in Mechanical Engineering from University of Petroleum, Shangdong, China, in May 1993; Received Master of Science degree in Petroleum Engineering from University of Petroleum, Beijing, China, in May 1996; completed the requirements for Master of Science degree in Computer Science at the Computer Science Department at Oklahoma State University in December 2002.

Professional Experiences: Teaching Assistant, Computer Science Department, Oklahoma State University, January 2001 to December 2001; Research Assistant, Computer Science Department, Oklahoma State University, June 2001 to July 2001.